HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

Architekturkonzept für Order Management Systeme mit integrierter Prozess-Engine

Martin Bröckl

Konstanz, 05.02.2014

BACHELORARBEIT

Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

an der

Hochschule Konstanz

Technik, Wirtschaft und Gestaltung

Fakultät Informatik

Studiengang Wirtschaftsinformatik

Thema: **Architekturkonzept für Order Management Systeme mit integrierter Prozess-Engine**

Bachelorkandidat: Martin Bröckl, Hauptstrasse 121, 8280 Kreuzlingen, Schweiz

1. Prüfer: Prof. Dr. Christian Johner
2. Prüfer: Dr. Beat Liver
Hagenholzstrasse 20/22, 8050 Zürich, Schweiz

Ausgabedatum: 05.11.2013
Abgabedatum: 05.02.2014

Zusammenfassung (Abstract)

Thema:	Architekturkonzept für Order Management Systeme mit integrierter Prozess-Engine
Bachelorkandidat:	Martin Bröckl
Firma:	Credit Suisse
Betreuer:	Prof. Dr. Christian Johner Dr. Beat Liver
Abgabedatum:	05.02.2014
Schlagworte:	Auftragsverwaltung, Architektur, BPM Plattform, BPM, BPMN 2.0, Prozess-Engine, Activiti

Das Ziel dieser Bachelorarbeit ist die Entwicklung eines Architekturkonzepts für ein Order Management System mit einer integrieren Prozess-Engine. Dabei soll es möglichst einfach sein neue Order-Typen in das Order Management System zu integrieren und mit Hilfe der Prozess-Engine auszuführen. Seitens der Credit Suisse wurden dazu die Systemanforderungen, sowie ein generischer Order-Life-Cycle vorgegeben. Unter Berücksichtigung dieser Aspekte wurde dann das Architekturkonzept entwickelt und mit Hilfe eines kleinen Prototyps verifiziert.

Zu Beginn wurde ein BPMN 2.0 Beispielprozess definiert. Daraus wurde dann, zusammen mit dem vorgegebenen Order-Life-Cycle ein generisches und modulares Prozess-Template entwickelt. Dieses dient der Modularisierung und der einheitlichen Verarbeitung von (Order-Typ)-Prozessen. Basierend darauf wurde ein Anwendungskonzept entwickelt, welches die Grundlage für den standardisierten Entwurf (Blueprint) hinsichtlich der Anwendungsklasse (Order Management) liefert. Abschließend wurde die Anwendungsarchitektur mit Hilfe eines kleinen Prototypen verifiziert.

Ehrenwörtliche Erklärung

Hiermit erkläre ich, Martin Bröckl, geboren am 13.12.1987,

(1) dass ich meine Bachelorarbeit mit dem Titel:

„Architekturkonzept für Order Management Systeme mit integrierter Prozess-Engine“

bei der Credit Suisse unter Anleitung von Professor Christian Johner und Beat Liver selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als die angeführten Hilfen benutzt habe;

(2) dass ich die Übernahme wörtlicher Zitate, von Tabellen, Zeichnungen, Bildern und Programmen aus der Literatur oder anderen Quellen (Internet) sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Konstanz, 05.02.2014

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

0 Executive Summary

Um die Verarbeitung eines Auftrags *(Order)* weitestgehend zu automatisieren und gleichzeitig flexibel zu gestalten, kann ein Order Management System *(OMS)* eine Business Process Management *(BPM)* Plattform einsetzen. Eine BPM Plattform ermöglicht es den Verarbeitungsverlauf von Orders als ausführbare Prozesse zu modellieren, diese zu Verwalten und in Verbindung mit einem Order Management System automatisiert ausführen zu lassen. Die Ausführung und Verwaltung von Prozessen wird dabei durch den Kern der BPM Plattform, der Prozess-Engine, ermöglicht. Derzeitige Order Management Systeme der Credit Suisse, verwenden dafür eine zentrale Plattform. Auch durchläuft jede Order während der Verarbeitung einen für diese Arbeit vorgegebenen Lebenszyklus *(Order-Life-Cycle)*, welcher den aktuellen Verarbeitungsstatus einer Order darstellt. In Bezug auf diese Thematik stellt sich die Frage, wie die Anwendungsarchitektur für ein Order Management System beschaffen sein muss, welches eine integrierte und somit dezentrale Prozess-Engine besitzt. Auch soll der Aufwand zur Integration und Automatisierung eines (Order)-Prozesses möglichst gering gehalten werden. Zusätzlich sollen mögliche Zusammenhänge zwischen dem Prozess und dem Life-Cycle einer Order untersucht werden. Ausgehend davon, ist das Ziel dieser Arbeit die Entwicklung eines standardisierten Entwurfskonzepts für die zukünftige Umsetzung von Order Management Systemen mit einer integrierten BPM Plattform.

Zu Beginn wird ermittelt, in welchem Verhältnis Prozess und Life-Cycle einer Order zueinander stehen können. Dabei wird auch gezeigt wie sich (Order)-Prozesse in einzelne Module unterteilen lassen, wodurch die parallele Entwicklung und Implementierung durch mehrere Entwickler ermöglicht wird. Anschließend wird die Standard-Anwendungsarchitektur des Order Management Systems erarbeitet. Bei der Beschreibung wird dabei zwischen technischen und individuellen Komponenten unterschieden. Damit soll gezeigt werden, welche Komponenten für unterschiedliche Implementationen des Order Management Systems gleichbleibend sind und welche für jedes System verschieden sind. Abschließend wird die Anwendungsarchitektur mittels eines Prototypen verifiziert.

Das Endergebnis der Arbeit zeigt, welcher Aufwand betrieben werden muss um einen neuen (Order)-Prozess in das OMS zu integrieren. Dabei wird die entwickelte Anwendungsarchitektur zur Verifikation auch den gegebenen Systemanforderungen gegenübergestellt. Danach wird der hier entwickelte dezentrale Ansatz in Bezug auf eine Prozess-Engine, einem zentralen Ansatz gegenübergestellt. Abschließend werden Schlussfolgerungen getroffen und ein Ausblick für die weitere Bearbeitung des Themas gegeben.

1 Einleitung

Das Order Management (Auftragsverwaltung) der Credit Suisse und der Finanzindustrie im Allgemeinen, umfasst eine Klasse von Anwendungen, um Aufträge, wie z.B. Zahlungsaufträge, Börsenaufträge, Aufträge zum Kauf und Verkauf von außerbörslich gehandelten Titeln, Devisen etc., verarbeiten zu können. Rechtlich muss man jedoch zwischen einem Auftrag *(Order)* und einem Geschäft *(*Trade*)* unterscheiden. In der Praxis, wird das Verwalten bzw. Verarbeiten von Aufträgen und Geschäften, auch als *Order- und Trade-Management* bezeichnet. Aus Sicht der technischen Umsetzung, kann man jedoch auf eine Unterscheidung der beiden Begriffe verzichten. Im Folgenenden wird daher nur der Begriff Order bzw. Order-Management verwendet.

Eine Order durchläuft während der Verarbeitung verschiedene Zustände, welche durch den sogenannten Auftrags-Lebenszyklus *(Order-Life-Cycle)* beschrieben sind. Der Order-Life-Cycle gibt beispielsweise an, wann eine Order aktiv ist oder ob diese vom Auftraggeber storniert wurde. Zudem wird jede Order auf eine bestimmte Weise verarbeitet, wobei sich der genaue Verlauf der Auftragsverarbeitung (Orderverarbeitung) im Vorfeld definieren und als Geschäftsprozess darstellen lässt. Dabei gibt es bezüglich der Darstellung unterschiedliche textuelle- und grafische Beschreibungsmethoden. Dies kann dabei helfen, die Prozesse eines Unternehmens besser zu verstehen und sie, den Veränderungen der Unternehmung entsprechend, anzupassen. Die Modellierung von Geschäftsprozessen bietet zudem die Möglichkeit, Informationssysteme flexibel auf die Bedürfnisse eines Unternehmens abzustimmen, indem die Systeme die modellierten Prozesse verwenden, um den damit verbundenen Arbeitsablauf *(Workflow)* zu automatisieren und zu steuern (Ould, 2005). Ein solches System ist zudem Teil der Anwendungsklasse „Process Aware Information Systems“, kurz *PAIS* (Reichert & Weber, 2012). PAISs entstanden aus der Notwendigkeit moderner Unternehmen heraus, sich an Änderungen des Umfelds flexibel und schnell anzupassen zu können. Um dies zu erreichen, kann ein solches Informationssystem beispielsweise eine Workflow- und Business Process Management (BPM) Plattform verwenden. Die Ausführung eines Geschäftsprozesses wird dabei von einer sogenannten Workflow-Engine oder auch Prozess-Engine gesteuert.

Mit dieser Arbeit soll der Frage nachgegangen werden, wie eine generische Architektur für ein Order Management System (kurz *OMS*) aussieht, welches eine in die Applikation eingebettete Prozess-Engine besitzt. Auch soll geklärt werden, wo genau die Trennung zwischen dem Life-Cycle einer Order und dem jeweiligen Geschäftsprozess liegt und in welchem Bezug diese zueinander stehen. Zusätzlich stellt sich die Frage, wie die Architektur beschaffen sein muss, um Geschäftsprozesse einfach in das OMS zu integrieren und ausführbar ma-

chen zu können. Dabei soll auch auf die Trennung von Prozess-Logik und Business-Logik geachtet werden. Daraus lässt sich die folgende wissenschaftliche Fragestellung ableiten:

Mit welcher Architektur für ein Order Management System können verschiedene Auftrags-Typen aus dem Devisenhandel, Zahlungsverkehr und so weiter, mit möglichst wenig Aufwand in das System integriert werden und unter Verwendung einer in das System eingebetteten Prozess-Engine ausführbar gemacht werden?

Das Ziel der Arbeit ist eine Applikations-Architektur, welche die wissenschaftliche Fragestellung beantwortet und zudem darstellt, in welchem Bezug der Life-Cycle einer Order und der zugehörige Geschäftsprozess zueinander steht. Dabei ist zu beachten, dass diese Arbeit ein Beispiel dafür sein soll, wie zukünftige Order Management Systeme aufgebaut sein könnten und mittels welcher Technologien diese entwickelt werden könnten. Aus praktischer Sicht bedeutet das, dass diese Arbeit ein Beispiel sowie die Grundlagen für den standardisierten und einheitlichen Entwurf *(Blueprint)* hinsichtlich der Anwendungsklasse liefern soll.

Zu Beginn wird die aktuelle Ausgangssituation aus Sicht der Credit Suisse erläutert. Anschließend werden die Randbedingungen für diese Arbeit aufgezeigt, sowie die Methodik, welche zur Beantwortung der wissenschaftlichen Fragestellung verwendet wird. Diese Punkte bilden den Grundstein für die Umsetzung dieser wissenschaftlichen Arbeit. Danach werden Order-Life-Cycle sowie die verwendete Workflow- und BPM-Plattform mit den zugehörigen Geschäftsprozessen genauer erläutert. Dabei wird beschrieben, in welchem Verhältnis Life-Cycle und Prozess zueinander stehen können und wie die zuvor erwähnte Prozess-Engine in das OMS integriert werden kann. Nach der Beschreibung der Architektur, wird diese mit Hilfe eines Prototyps verifiziert, wobei auch auf die verwendeten Technologien eingegangen wird. Danach wird das Ergebnis der Arbeit präsentiert und es werden gesicherte Aussagen getroffen. Abschließend werden Schlussfolgerungen erläutert und ein Ausblick gegeben.

2 Ausgangssituation

Unternehmen sind zunehmend daran interessiert Informationssysteme so zu gestalten, dass die Geschäftsprozesse der Unternehmung im Mittelpunkt stehen. Dabei besteht die zentrale Aufgabe der Geschäftsanwendungen darin, die richtigen Geschäftsfunktionen mit den benötigten Informationen zur Verfügung zu stellen und gleichzeitig zum richtigen Zeitpunkt an den richtigen Benutzer weiterzuleiten. Zusammen mit diesem Trend hat sich eine neue Generation von Informationssystemen entwickelt, Process Aware Information Systems *(PAIS)* (Reichert & Weber, 2012). Beispiele für PAISs sind *Workflow Management Systeme*, *Case Handling Tools* und *Service Orchestration Engines.* Aus architektonischer Sicht, separiert ein PAIS Prozesslogik (unterschiedlicher Art) von funktionaler Anwendungslogik (z.B. Berechnungen), woraus sich eine zusätzliche Schicht in Bezug auf die Architektur einer Applikation ergibt. Der Kern dieser Schicht besteht meist aus einem Prozess Management System, womit sich Geschäftsprozesse modellieren, ausführen, überwachen und steuern lassen (Reichert & Weber, 2012). Grundsätzlich unterscheidet man zwischen zwei unterschiedlichen Typen von Prozessen. Zum einen gibt es Prozesse, deren vollständige Logik schon vor der eigentlichen Ausführung bekannt ist und demnach vorab modelliert werden können. Diese Prozesse werden als vordefinierte Prozesse bezeichnet. Zum anderen gibt es wissensintensive Prozesse, deren vollständige Logik häufig erst während der Ausführung ersichtlich wird. Ein weiteres Merkmal vordefinierter Prozesse ist die Wiederholbarkeit, da im Gegensatz zu wissensintensiven Prozessen der genaue Verlauf der Verarbeitung im Voraus bekannt ist (Reichert & Weber, 2012). Im Rahmen dieser Arbeit wird die Architektur für ein OMS entwickelt, die sich auf die Verarbeitung von vordefinierten und aktivitätsbasierten Geschäftsprozessen konzentriert. Aktivitätsorientiert bedeutet, dass ein Prozess in mehrere einzelne Aktivitäten *(Tasks)* unterteilt ist. Jeder dieser Tasks führt dann einen atomaren und logisch zusammenhängenden Prozessschritt durch. Beispiele für aktivitätsorientierte Prozessbeschreibungssprachen sind die Business Process Management Notation 2.0 *(BPMN 2.0)* und die Business Process Execution Language *(BPEL)*, wobei im Rahmen dieser Arbeit Prozesse unter Verwendung von BPMN 2.0 modelliert werden. Im weiteren Verlauf dieser Arbeit, wird ein in BPMN modellierter, vordefinierter und aktivitätsbasierter Geschäftsprozess zu Vereinfachung nur noch als *Prozess* bezeichnet.

Prozessklassen vordefinierter Prozesse in Bezug auf die Credit Suisse sind z.B. Zahlungsaufträge oder auch Börsenaufträge, wobei der Fokus der Thesis auf Orders in Bezug auf standardisierte Finanzkontrakte und Dienstleistungen, wie z.B. Zahlungsaufträge, Devisengeschäfte usw., liegt. Die einzelnen Prozesse innerhalb dieser Klassen stellen vordefinierte Standardprozesse dar und sind beliebig oft anwendbar. Auch ist eine weitgehendste Auto-

matisierung notwendig, um eine hohe Anzahl von Aufträgen kostengünstig verarbeiten zu können. Eine Untersuchung in Bezug auf die Verarbeitung von Prozessen anderer Prozessklassen, wie z.B. *strukturierter Produkte* (SVSP, 2014) mit Hilfe des OMS, ist nicht Bestandteil dieser Arbeit.

PAISs können aus unterschiedlichen Blickwinkeln betrachtet werden. Dazu zählen *Funktionsorientierte*, *Verhaltensorientierte*, *Informationsorientierte*, *Organisationsorientierte*, *Operationsorientierte* und *Zeitorientierte* Betrachtungsweisen von Prozessmodellen (Reichert & Weber, 2012). Im Folgenden wird auf drei, für diese Arbeit relevante Gesichtspunkte, genauer eingegangen:

Funktionsorientierte Betrachtungsweise: Diese Sicht liefert Informationen über die einzelnen Aufgabenteile, bzw. Tasks, eines Prozesses. Dazu zählt beispielsweise, ob ein einzelner Task von einer Person oder einem System ausgeführt wird, oder ob dieser einen Subprozess darstellt (Reichert & Weber, 2012, S. 21-22).

Verhaltensorientierte Betrachtungsweise: Diese Sicht liefert Informationen über den Kotrollfluss zwischen einzelnen Tasks eines Prozesses. Der Kontrollfluss gibt beispielsweise Aufschluss über die Reihenfolge der Tasks und welche Bedingungen, mit der Durchführung eines Tasks verbunden sind (Reichert & Weber, 2012, S. 22-25).

Operationsorientierte Betrachtungsweise: Diese Sicht stellt die Implementation einzelner Tasks eines Prozesses dar. Im Falle der zuvor besprochenen Prozesse wäre dies die Implementation einer Funktion, die ausgeführt wird, wenn ein Task durchgeführt wird. Ein Beispiel dafür wäre der Aufruf einer lokalen Methode oder eines Services, welcher von einem externen System angeboten wird (Reichert & Weber, 2012, S. 27-29).

Derzeit wird innerhalb der Credit Suisse für die Automatisierung von Geschäftsprozessen eine zentrale Workflow- und Business Process Management *(BPM)* Plattform eingesetzt. Zentral bedeutet in diesem Zusammenhang, dass Anwendungen den zu verarbeitenden Geschäftsprozess auf die zentrale BPM Plattform auslagern. Zur Verarbeitung der Prozesse werden dann wiederum die Services der jeweiligen Anwendung von der BPM Plattform aufgerufen. Bei der Plattform handelt es sich um die Oracle BPM Suite 11g, welche in einem internen Projekt in die Credit Suisse integriert wurde. Damit ein menschlicher Benutzer *(User)* während der Verarbeitung eines Prozesses mit diesem interagieren kann, stellt die zentrale BPM Plattform Usern eine separate Komponente zur Verfügung. Diese wird als Arbeitsliste *(Worklist)* bezeichnet. Ein prozessbeteiligter User, kann somit die ihm zugeteilten Prozessaktivitäten verwalten und erledigen. Im Rahmen dieser Arbeit, wird auf diese Komponente jedoch nicht weiter eingegangen. Dabei wird vorausgesetzt, dass für derartige Akti-

vitäten eine zentrale Worklist zur Verfügung steht und diese über Services in die hier entwickelte Anwendungsarchitektur für ein OMS angebunden werden kann.

Momentan sind diverse OMSs, basierend auf der zentralen BPM Plattform, im Einsatz. Entgegen dem zentralisierten Ansatz, wird in dieser Arbeit ein dezentraler Ansatz erarbeitet. Dabei soll neben der Anwendungs-Logik eines OMS, auch die Prozess-Logik innerhalb desselben Web-Application-Servers (kurz *WAS*) zur Verfügung gestellt werden. Der Hauptvorteil dieser Variante ist, dass selbst bei weniger komplexen Anwendungen eine funktionale Separierung von Prozessbezogener- und Anwendungsbezogener Logik stattfindet.

2.1 Order Lebenszyklus (Order Life-Cycle)

Wie bereits erwähnt durchläuft jede Order im Laufe der Verarbeitung verschiedene Zustände. Beschrieben wird dies durch den Order Life-Cycle (Zustandsautomat). Die verschiedenen Zustände des Life-Cycles stellen die aktuelle Phase der Verarbeitung dar, in der sich eine Order befindet. Ein Order-spezifischer Prozess, welcher zur Orderverarbeitung durchlaufen wird, korrespondiert dabei an bestimmten Stellen mit dem Order-Life-Cycle. Wichtig hierbei ist, dass der Life-Cycle Status einer Order während der Ausführung eines Geschäftsprozesses verändert wird. Auf diese Punkte wird in einem späteren Kapitel genauer eingegangen. Für diese Arbeit wurde der Order Life-Cycle seitens der Credit Suisse vorgegeben und ist in Abbildung 1 dargestellt.

Nach dem Erzeugen einer neuen Order, kann entweder der Status „Pending“ oder „Active“ angenommen werden. Erst wenn eine Order den Zustand „Active“ besitzt, können die Zustände „Cancel“, „Mature“ oder „Replace“ folgen. Die Zustände werden im Folgenden genauer beschrieben.

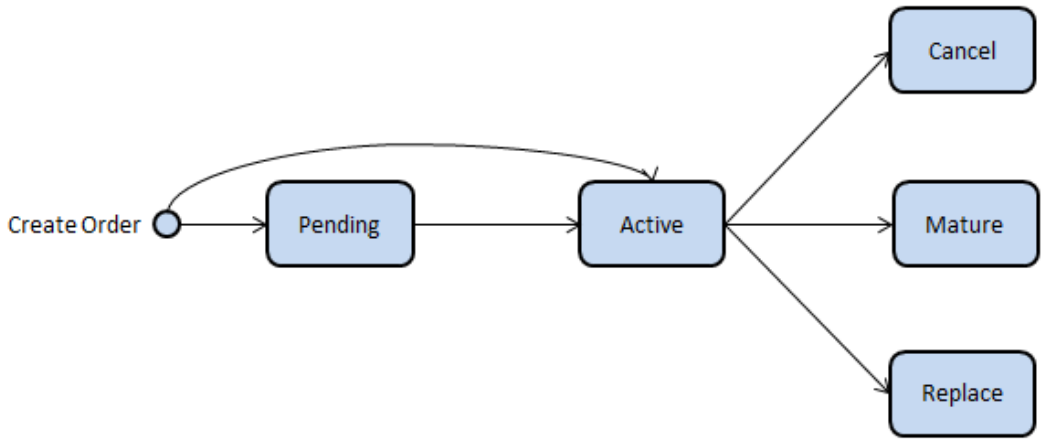


Abbildung 1: Order-Life-Cycle

Pending: Bedeutet, dass eine Order unvollständig ist und dass noch auf weitere Informationen gewartet wird. Aus Sicht aktivitätsorientierter Prozessmodelle kann das bedeuten, dass Prozessaktivitäten für die Erstellung einer Order noch ausstehend sind und es dennoch notwendig ist, dass das Geschäft als solches schon bekannt ist. Fehlende Informationen können von anderen Systemen oder aber auch Personen kommen, die an dem Prozess beteiligt sind.

Active: Bedeutet, dass sich eine Order in der Durchführungsphase befindet. Ein Kunde könnte beispielsweise eine Zahlung beauftragen, welche an einem bestimmten Termin erfolgen soll. Der Status „Active" besteht in diesem Fall solange bis der Termin eingetroffen ist und die beauftragte Zahlung abgewickelt werden kann. Zudem besitzt der Kunde die Möglichkeit, die Order vor Eintritt des Termins zu ändern oder zu stornieren.

Mature: Bedeutet, dass die Verarbeitung der Order abgeschlossen ist und die Zahlung *(Settlement)* der Order eingeleitet wurde.

Cancel: Bedeutet, dass die Durchführung der Order gestoppt wurde und alle notwendigen Schritte eingeleitet wurden um die Verarbeitung der Order abzubrechen bzw. rückgängig zu machen.

Replace: Bedeutet, dass eine Order durch eine oder mehrere Nachfolgeorder(s) ersetzt und die Verarbeitung der aktuellen Order daher beendet ist. Der ursprüngliche Prozessfall wird durch die Verarbeitung der Nachfolgeorder(s) weiter bearbeitet. Beispielsweise könnte ein Trader der Bank den Auftrag haben, für insgesamt zehn Kunden jeweils 1.000.000 Amerikanische Dollar gegen Schweizer Franken zu kaufen. An dieser Stelle könnte der Trader dann einen sogenannten „Block-Trade" initiieren. Dabei würde eine Order für den Kauf von 10.000.000 Dollar gegen Schweizer Franken, in Abhängigkeit eines gewünschten Termins *(Valuta-Datum)* aufgegeben werden. Diese würde dann mit einem „Replace" für die zehn Kunden, in insgesamt zehn Nachfolgeorders zu je 1.000.000, aufgeteilt werden. Dabei hat jede einzelne Order eine Referenz auf die ursprüngliche Order. Auch würden alle Nachfolgeorders mit dem gleichen Preis *(Devisenkurs)* bzw. Wechselkurz gehandelt werden. Dieser wird durch den Kauf der 100.000.000 Dollar, in Abhängigkeit des zu diesem Zeitpunkt aktuellen (SPOT)-Marktwerts, festgelegt.

2.2 FX-Spot Order – Grundlegende Merkmale

Für die Entwicklung der Architektur des OMSs, wird eine Beispiel-Order aus dem Devisenhandel, eine sogenannte FX-Spot Order (Foreign Exchange Spot Order), zur Hilfe genommen. Damit soll die Beantwortung der wissenschaftlichen Fragestellung vereinfacht und dem

Leser ein praxisnahes sowie leicht verständliches Beispiel geliefert werden. In Bezug auf den Prototyp wird nur ein Teil der Beispiel-Order implementiert, worauf im entsprechenden Kapitel näher eingegangen wird. Im Folgenden werden die wichtigsten Merkmale einer FX-Spot Order, also des Order-Typs, erläutert. Im weiteren Verlauf der Arbeit, wird das Beispiel dann mit Hilfe der BPMN 2.0 Notation als Geschäftsprozess modelliert und zur Implementation des Prototyps verwendet.

Eine FX-Spot Order ist eine Vereinbarung zwischen zwei Parteien, um einen bestimmten Betrag einer Währung zu verkaufen/kaufen und dafür einen bestimmten Betrag einer anderen Währung zu kaufen/verkaufen. Dabei stellt der aktuelle Kurs *(Quote)* des jeweiligen Währungspaars den Preis einer Währung, ausgedrückt in einer anderen Währung, dar. Wird der angebotene Kurs vom Kunden akzeptiert, so wird der eigentliche Geldtransfer (Settlement) in der Regel zwei Tage nach dem Kauf initiiert. Abweichungen werden der Einfachheit halber an dieser Stelle nicht weiter berücksichtigt.

Eine FX-Spot Order lässt sich als ein Geschäftsprozess modellieren. Dabei ist der vollständige Ablauf des Prozesses im Voraus bekannt und zählt somit zur Klasse der vordefinierten, aktivitätsorientierten und wiederholbaren Prozesse. Aus Sicht der Credit Suisse, sind an einem FX-Spot Order Geschäftsprozess verschiedene Informationssysteme beteiligt. Dazu zählen beispielsweise *Preis-Engines*, *Position-Keeping-Systeme*, *Zahlungssysteme* etc. Wie zuvor beschrieben, besteht ein aktivitätsbasierter Prozess aus einzelnen Aktivitäten *(Tasks)*. Zur Ausführung eines Tasks können neben internen Funktionen des OMS, also desselben Systems, auch externe Funktionen anderer Systeme verwendet werden, was durch die Operationsorientierte Betrachtungsweise dargestellt werden kann. Externe Funktionen werden dabei mit Hilfe von Servie-Orientierte-Architektur *(SOA)* Services angesprochen. Beispiele für einzelne Tasks sind das Anfragen des Kurses für ein bestimmtes Währungspaar, das Anfragen von Auftraggeber-spezifischen Informationen, das Analysieren von Risiken, etc.

3 Randbedingungen und Methodik

Im Folgenden werden die Randbedingungen in Verbindung mit diesem Thesis-Projekt erläutert. Dabei wird auch auf die funktionalen Systemanforderungen eingegangen, die an das OMS gestellt sind. Danach wird die Methodik beschrieben, mit der die wissenschaftliche Fragestellung beantwortet wird.

3.1 Randbedingungen

Das OMS soll eine Workflow - und BPM Plattform zur Ausführung vordefinierter und aktivitätsorientierter Geschäftsprozesse verwenden. Der Kern der Plattform, welcher die Prozesse ausführt, wird als *Prozess-Engine* bezeichnet. Für die Umsetzung der Arbeit ist es notwendig, eine Prozess-Engine zu verwenden, die sich in das System integrieren lässt. Konkret bedeutet dies, dass die Engine ein API zur Verfügung stellen muss, welche unter Verwendung der Zielsprache des OMS verwendet werden kann. Außerdem ist es vorgesehen, eine Engine einzusetzen, welche die Möglichkeit besitzt, BPMN 2.0 Prozesse auszuführen. Andere Modellierungsmethoden, wie z.B. BPEL, wären allerdings ebenfalls denkbar. Der Vorteil des Einsatzes einer integrierten Prozess-Engine liegt zum einen in der einheitlichen Implementierungsumgebung und zum anderen im einfacheren Systembetrieb (Signavio, 2012). Eine weitere Bedingung ist, dass das OMS Services für das Erzeugen *(Create)*, Lesen *(Read)*, Ersetzen *(Update)* und Löschen *(Delete)* von Orders zur Verfügung stellt, die im weiteren Verlauf als *CRUD-Services* bezeichnet werden. Diese Services sind nach außen die einzige Methode, um mit dem OMS zu kommunizieren. Der Prototyp, mit dem der theoretische Entwurf verifiziert werden soll, implementiert aus zeitlichen Gründen nur das Erzeugen einer Order, den Create-Service. Auch ist zu beachten, dass im Rahmen der Thesis kein Produktivsystem entworfen bzw. entwickelt werden soll, sondern lediglich ein fundiertes Beispiel für weitere Diskussionen gegeben werden soll. Aufgrund des begrenzten zeitlichen Rahmens dieser Arbeit von drei Monaten, wird zur Reduzierung der Komplexität der Implementation, eine Open-Source BPM Plattform eingesetzt (siehe 4.2). Diese wird dann in das hier entwickelte OMS eingebettet und über einen Web-Applikations-Server *(WAS)* zur Verfügung gestellt. Diese Vorgehensweise dient lediglich der Entwicklung eines standardisierten Entwurfs. In einem Folgeprojekt, könnte gezeigt werden, wie die hier verwendete Open-Source BPM Plattform mit der kommerziellen Lösung der Credit Suisse ersetzt werden kann.

Systemanforderungen

An das hier zu entwickelnde OMS werden verschiedene funktionale Systemanforderungen gestellt. Nicht-funktionale Anforderungen werden gemäß der Disposition, der Einfachheit halber, nicht weiter berücksichtigt. Im Folgenden sind die Anforderungen aufgelistet, welche seitens der Credit Suisse für diese Arbeit vorgegeben wurden. Um deren Umsetzung im weiteren Textverlauf besser nachvollziehen zu können, werden diese an den jeweiligen Stellen referenziert.

- Architekturentwurf des Order Management Systems (OMS) basierend auf einer Service-Orientierten-Architektur (SOA), d.h. das OMS bietet CRUD-Services an, um eine Order zu initialisieren bzw. zu manipulieren.
- Das OMS verwendet unter anderem Services anderer Systeme für die Implementierung von Prozessaktivitäten
- Für die Ausführung von Prozessaktivitäten, verwendet das OMS neben externen Service-Aufrufen auch interne Funktionen sowie Benutzerinteraktionen.
- Das OMS soll modular erweiterbar sein, d.h., das System wird zur besseren Erweiterbarkeit in einzelne Services unterteilt. Erweiterbar bedeutet in diesem Zusammenhang folgendes:
- Neue Services (lokal, remote) und neue Order-Typen (neue Prozesse) können mit möglichst wenig Aufwand in das System integriert werden.
- Funktionen des OMS sind versionierbar, d.h., dass beispielsweise ein Create einer Order, für verschiedene Order-Typen unterschiedliche Implementierungen haben kann.
- Bereits vorhandene Prozesse können möglichst leicht gewartet werden
- Implementierung des Systems unter Verwendung einer BPMN 2.0 Workflow-Engine, welche in das System integriert werden soll.
- Für den Anwendungsentwickler soll es auf einfache Art möglich sein, Prozess-Aspekte von der restlichen Anwendungslogik zu trennen.
- Die Architektur soll die Möglichkeit bieten, neue Order-Typen so implementieren zu können, o dass theoretisch mehrere Entwickler gleichzeitig an verschiedenen Aufgaben arbeiten können.

3.2 Methodik

Unter Verwendung eines FX-Spot Order Beispielprozesses wird zu allererst der theoretische Entwurf des OMS entwickelt. Dabei wird auch auf die fachlichen Zusammenhänge zwischen Order Life-Cycle (siehe 2.1) und Geschäftsprozess eingegangen, welche mit in das Architekturkonzept einfließen sollen. Das Resultat ist dann eine Anwendungsarchitektur, welche die Funktionsweisen aller Komponenten der Architektur darstellt und erläutert. Danach wird unter Verwendung der Anwendungsarchitektur die Technische Architektur entwickelt. Hierbei werden den jeweiligen Komponenten, Technologien zugewiesen, mit denen die Architektur implementiert werden kann. Parallel zur Architektur werden die Schnittstellen des Order Management Systems definiert. Diese Vorgehensweise lehnt sich an eine Methode zur Beschreibung einer Architektur mit dem Namen „Quasar“ an, welche von der Firma sd&m entwickelt wurde (Siedersleben, 2003). Anschließend wird die Architektur durch die Implementation eines Prototyps verifiziert. Dabei ist zu beachten, dass aus zeitlichen Gründen keine vollständige Implementierung durchgeführt werden kann. Der genaue Umfang ist in dem jeweiligen Kapitel definiert und begründet. Das Methodische Vorgehen ermöglicht, ein konzeptionelles bzw. theoretisches Ergebnis zu liefern, welches dann mit Hilfe des Prototyps praktisch verifiziert werden kann.

4 Durchführung

Dieses Kapitel beinhaltet die praktische Durchführung der Arbeit. Dabei wird die in 3.2 beschriebene Methodik angewandt. Am Ende wird das Ergebnis der Arbeit detailliert beschrieben und unter Verwendung des Prototyps verifiziert.

4.1 Wichtige Begrifflichkeiten und deren gegenseitige Abgrenzung

Um den weiteren Verlauf des Dokuments besser verstehen zu können, ist es notwendig, wichtige Begriffe in Verbindung mit dem OMS näher zu erläutern. Dabei werden die Begriffe miteinander in Verbindung gesetzt bzw. voneinander abgegrenzt.

4.1.1 Service

Den Systemanforderungen zufolge, soll das OMS Nutzern des Systems, ob Mensch oder Maschine, SOA-Services zur Verfügung stellen (siehe 3.1). Diese ermöglichen es, eine Order zu erteilen und diese während der Durchführung zu beeinflussen. Dazu bietet das OMS CRUD-Services an, welche von außen die einzige Möglichkeit darstellen, um mit dem OMS zu kommunizieren. Jeder dieser Services (Create, Read, Update und Delete) kann im Sinne einer SOA mit Hilfe unterschiedlicher Technologien realisiert werden. Beispiele dafür sind Representational State Transfer *(REST) Services*, WebServices, Java Message Service *(JMS)* Services, etc. Wurde eine Order von einem Client erteilt (Create), hat dieser die Möglichkeit, neben dem Lesen (Read) von Orders, diese auch zu ändern (Update) oder zu löschen (Delete). Ist die Ausführung eines Services fehlerhaft, so erhält der Client eine entsprechende Fehlermeldung.

4.1.2 Prozess

Ein Prozess stellt den logischen Verlauf einer Orderverarbeitung dar, welcher mit Hilfe diverser Prozessbeschreibungssprachen modelliert werden kann. Innerhalb dieses Dokuments wird dazu ausschließlich BPMN 2.0 zur Modellierung verwendet. In einem BPMN 2.0 Prozess wird der Prozessverlauf durch die Prozessfluss-Logik und der Anordnung der einzelnen Aktivitäten (Tasks) bestimmt. Die Prozessfluss-Logik gibt beispielsweise an, ob Aktivitäten sequenziell oder parallel ausgeführt werden. Auch kann die Ausführung von Aktivitäten verschiedenen Bedingungen vorausgesetzt sein, um nur einige Beispiele zu nennen. Ein Prozess kann zudem aus mehreren einzelnen (Teil)-Prozessen bestehen, welche in der Summe die vollständige Logik des gesamten Prozesses darstellen. Von einem modellierten Prozess können dann mit Hilfe einer Prozess-Engine beliebig viele Prozess-Instanzen erzeugt werden. Jede Instanz besitzt dabei ein eigenes (Daten)-Objekt, dass gleichzeig mit dem Erzeugen der jeweiligen Prozess-Instanz erstellt wird (Reichert & Weber, 2012). Der aktuelle Sta-

tus einer Prozess-Instanz wird von der Prozess-Engine verwaltet und in einer eigenen Datenbank persistiert. Wichtig ist hier, dass der Status des Prozesses von dem Status einer Order, wie z.B. einer FX-Spot Order (siehe 2.2), verschieden ist. Während eine Order den in 2.1 gezeigten Order Life-Cycle durchläuft und vom OMS selbst verwaltet wird, wird der Status einer Prozess-Instanz von der Prozess-Engine gehandelt. Analogien von Order- und Prozess-Life-Cycle werden in 4.6.3 detailliert erläutert.

4.1.3 Objekt

Jeder Prozess besitzt ein eigenes (Daten)-Objekt, welches die minimale Summe der benötigten Informationen besitzt, die aus Sicht einer jeden Order gehalten werden müssen. Ein Beispiel für gehaltene Informationen wäre der aktuelle Status im Order Life-Cycle (siehe 2.1). Auch enthält das Objekt individuelle Attribute bezüglich des jeweiligen Finanzprodukts, welches mit einer zugehörigen Order bzw. einem Order-Prozess verarbeiten werden soll (z.B. gehandelter Betrag, Währungspaar, Devisenkurs, Valuta (SPOT), Handelstag, etc.). Mit jeder Instanz eines Prozesses, wird auch eine neue Instanz des jeweiligen Objektes erstellt. Während der Ausführung eines Prozesses steht das Objekt dann allen Aktivitäten zur Verfügung, wobei jede Aktivität ein oder mehrere Attributs-Werte des Objektes verändern kann. Diese Änderung steht dann allen darauf folgenden Aktivitäten zur Verfügung. An bestimmten Prozessabschnitten muss das Objekt persistiert werden, was über eine separate Komponente des OMS ermöglicht wird.

4.1.4 Life-Cycle

Während der Ausführung eines Order-spezifischen Prozesses (z.B. FX-Spot Order Prozess), wird der Order Life-Cycle durchlaufen. Dabei wird der aktuelle Status der Order von der jeweiligen Instanz des (Daten)-Objektes gehalten. Wichtig ist, dass der Life-Cycle für jede einzelne Instanz bis zu einem Endpunkt durchlaufen wird. Wird beispielsweise ein „Update“ einer Order über einen der CRUD-Services initiiert, so wird der Status des Objektes auf „Replaced“ gesetzt. Anschließend wird dann eine neue Prozess-Instanz desselben oder eines anderen Prozesses gestartet. Dabei wird gleichzeitig eine neue Objekt-Instanz, des zu einem Prozess gehörenden (Daten)-Objekts, erstellt. Die genaue Funktionsweise zur Verwaltung des Order-Life-Cycles, wird in 4.6 genauer beschrieben.

4.2 Workflow- und BPM-Plattform

Mit Hilfe einer Workflow und BPM-Plattform lassen sich Geschäftsprozesse modellieren und verwalten. Sie bietet zudem die Möglichkeit, die Prozesse durch ein Softwaresystem ausführbar zu machen und den Kontrollfluss des Prozesses automatisiert steuern zu lassen. Laut den Systemanforderungen (siehe 3.1) soll das OMS eine BPMN 2.0 Prozess-Engine

zur Ausführung von Prozessen verwenden. Für diese Arbeit wird eine Open-Source Workflow und Business Process Management (BPM) Plattform namens Activti (Activiti™, 2014) verwendet. Der Kern besteht aus einer BPMN 2.0 Prozess-Engine, die in Java geschrieben ist und ein API zur Verfügung stellt, mit der sich die Engine einfach in die eigene Java-Applikation integrieren lässt. Activiti kann innerhalb Java-Applikationen auf Servern, in einem Cluster oder in einer Cloud eingesetzt werden. Folgend werden die relevanten bzw. verwendeten Bestandteile der Plattform beschrieben.

4.2.1 Komponenten

Activiti besteht aus verschiedenen Komponenten, wobei der Kern, der für die Ausführung der Prozesse verantwortlich ist, von der Prozess-Engine gebildet wird. Neben der Engine stehen unter anderem diverse Tools für die Modellierung der Prozesse zur Verfügung und darüber hinaus ebenfalls Management-Tools für deren Verwaltung. Im Rahmen dieser Arbeit wird im Wesentlichen die Prozess-Engine verwendet. Zusätzlich wird zur Modellierung der Prozesse der Activiti Designer eingesetzt, welcher sich in die Entwicklungsumgebung Eclipse integrieren lässt. Abbildung 2 stellt die Komponenten der Plattform grafisch dar. Genauere Beschreibungen der Komponenten können der Activiti-Webseite sowie dem Acticiti User-Guide auf der Activiti-Website entnommen werden (Activiti™, 2014).

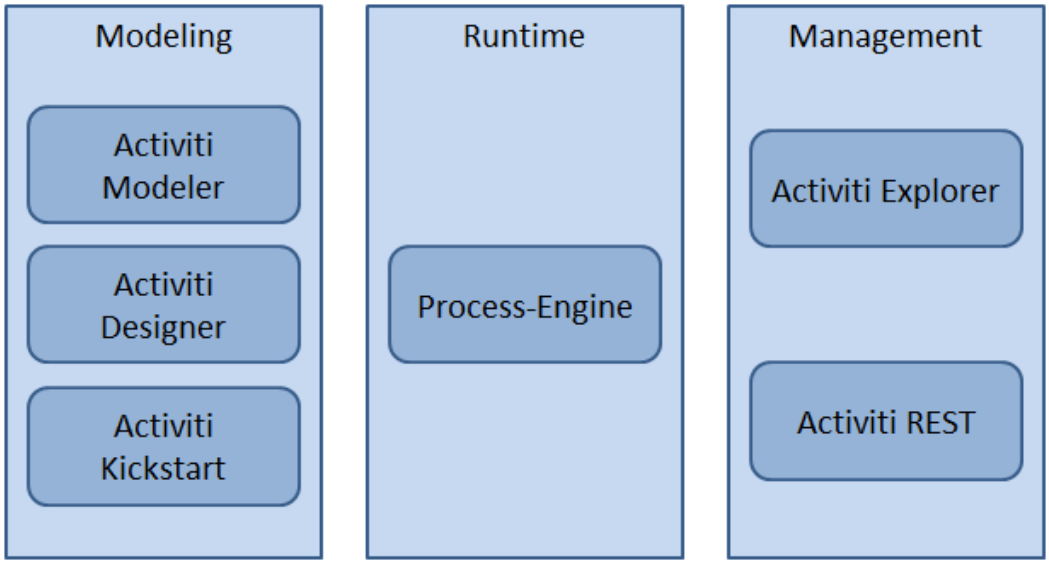


Abbildung 2: Activiti Komponenten

4.2.2 Process-Engine

Die Activiti Prozess Engine, führt standardmäßig BPMN 2.0 Prozesse aus. Dabei liefert Activiti eine Menge an vordefinierten BPMN 2.0 Aktivitäts-Typen. Diese lassen sich auch durch neue, eigene Aktivitäts-Typen erweitern. Zusätzlich gibt es die Möglichkeit neben BPMN 2.0,

noch weitere Prozessbeschreibungssprachen zu verwenden und in Activiti einzubinden. Mit der Engine lassen sich Instanzen von Prozessen bilden, ausführen, überwachen und verwalten. Wie zuvor erwähnt, besitzt die Prozess-Engine eine eigene Datenbank, welche auch ein eigenes Schema besitzt und von Activiti selbst definiert ist. Beispielsweise werden in der Datenbank modellierte Prozesse persistiert. Somit kennt die Prozess-Engine den genauen Verlauf jedes einzelnen Prozesses. Außerdem werden Informationen über laufende Instanzen der Prozesse gehalten und während der Ausführung verwaltet. Ein weiteres Feature der Engine, welches an dieser Stelle zu erwähnen ist, sind *Execution-Listener*. Diese können beispielsweise dazu verwendet werden, abhängig von bestimmten Events während der Ausführung eines Prozesses, Java-Code oder Scripts auszuführen. Das ist besonders hilfreich, wenn ein System Programm-Code ausführen soll, der nicht direkt mit dem Geschäftsprozess zusammenhängt und demnach nicht innerhalb des Prozesses modelliert werden soll. Diese Funktion der Engine ist für das Modellieren von Geschäftsprozessen, in Verbindung mit dem Life-Cycle einer Order, von großer Wichtigkeit (siehe 4.6).

In Activiti besitzt jeder Prozess neben der BPMN 2.0 Darstellung auch eine XML-Repräsentation. Diese wird unter Verwendung des Activiti Designers in Eclipse automatisch generiert. Der generierte XML-Code, wird dann von der Prozess-Engine verwendet, um die modellierten Prozesse ausführen zu können. Alternativ kann der XML-Code auch ohne Verwendung des Activiti Designers, d.h. von Hand implementiert werden.

4.2.3 Erstellen und Implementieren ausführbarer Prozesse

Bevor ein Geschäftsprozess implementiert werden kann, muss zuerst entschieden werden, welche Aktivitäten des Prozesses automatisiert werden sollen. Anschließend muss ein ausführbarer Geschäftsprozess, wie z.B. ein BPMN 2.0 Prozess, erstellt werden, der die zu automatisierenden Aktivitäten modelliert. Komplexe Aktivitäten werden dabei oft auch als Subprozesse modelliert. In der Regel geschieht das mit Hilfe eines Tools, wie z.B. dem Activiti Designer oder Activiti Modeler. Nachdem ein Prozess modelliert wurde, müssen den einzelnen Aktivitäten (Tasks) entsprechende Funktionen zugewiesen werden. Diese stellen dann den eigentlichen Programmcode dar, welcher bei der Ausführung einer Aktivität aufgerufen wird. Dabei kann es auch sein, dass User an einem Prozess beteiligt sind und dass zur Durchführung bestimmter Aktivitäten deren Interaktion benötigt wird. Bei der Modellierung eines Prozesses, kann dazu festgelegt werden, welche(r) bestimmte(r) Benutzergruppe bzw. Benutzer (User) die Prozessaktivität durchführen soll. Bezogen auf die Activiti Komponenten (siehe Abbildung 2), kann die Interaktion zwischen User und Prozess, mit Hilfe des Activiti Explorers realisiert werden. Diese Funktionalität bzw. Komponente wird jedoch innerhalb dieser Arbeit nicht weiter behandelt (siehe 2).

Nachdem ein ausführbarer BPMN 2.0 Prozess erstellt wurde, kann dieser mit Hilfe der Prozess-Engine in der Activiti Laufzeitumgebung bekannt gemacht werden. Das bedeutet, dass der modellierte Prozess in der Datenbank persistiert wird. Somit kennt die Prozess-Engine den genauen Verlauf eines Prozesses. Während der Ausführung eines Prozesses, steuert die Prozess-Engine den Life-Cycle der jeweiligen Prozess-Instanz, orchestriert die einzelnen Implementationen der Aktivitäten und Subprozesse, welche mit dem Prozess verbunden sind, etc. Beim Aufruf einer Aktivität, zusammen mit der damit verbundenen Implementation, werden auch die Ein- und Ausgabeparameter von der Prozess-Engine gehandelt. Diese Vorgehensweise ist nicht Activiti-spezifisch sondern funktioniert bei jeder BPM Plattform gleich. Dies wird in (Reichert & Weber, 2012, S. 30-33) näher beschrieben.

4.3 FX-Spot Order Beispielprozess

Um die in 2.2 erläuterte FX-Spot Order mit Hilfe der Activiti Prozess-Engine auszuführen, muss diese unter Verwendung der BPMN 2.0 Notation als ein Geschäftsprozess modelliert werden. Bei der Modellierung des Prozesses wurde weder auf die Vollständigkeit noch auf die hundertprozentige Richtigkeit des Prozesses Wert gelegt. Vielmehr soll ein einfacher und praxisnaher Beispielprozess modelliert werden, mit dessen Hilfe sich die funktionalen Anforderungen des OMS verwirklichen und veranschaulichen lassen. Die dabei entstandene Lösung soll dann auf andere BPMN 2.0 Geschäftsprozesse übertragen werden können. Mit der Modellierung von Geschäftsprozessen unter Verwendung der BPMN 2.0 Notation, wird auch die Funktionsorientierte- sowie Verhaltensorientierte Betrachtungsweise veranschaulicht (siehe 2). Nachfolgend wird der FX-Spot Order Geschäftsprozess, zunächst aus fachlicher Sicht, beschrieben. Anschließend wird dieser mit Hilfe von BPMN 2.0 modelliert, wodurch die technische Sicht darstellt wird.

4.3.1 Fachliche Sicht

Ein Kunde der Credit Suisse, möchte beispielsweise 1.000.000 Amerikanische Dollar gegen Schweizer Franken kaufen. Dazu fragt er das OMS nach dem aktuellen Kurs (Quote) des gewünschten Währungspaares. Wichtig ist, dass das OMS nicht selbst über die Quote verfügt, sondern diese lediglich mit Hilfe eines Services von der dafür zuständigen Preis-Engine anfordert und an den anfragenden Client zurückliefert. Nachdem der Kunde die Quote erhalten hat, kann er diese entweder akzeptieren oder aber, er fragt bei dem OMS eine neue Quote an. Akzeptiert er die Quote, so sendet er eine Anfrage mit allen notwendigen Daten für eine FX-Spot Order an das OMS. Übermittelte Daten sind z.B.: die Quote, die Höhe des Betrags, Kundeninformationen, die zu kaufende/verkaufende Währung etc. Nachdem das OMS die Anfrage erhalten hat, wird zuallererst die Gültigkeit der übermittelten Quote überprüft. Schlägt diese Prüfung fehl, d.h. die Quote ist ungültig, so wird der Prozess abgebro-

chen und der Kunde wird von OMS entsprechend informiert. Nach erfolgreicher Validierung der Quote, überprüft das OMS das jeweilige Kreditrisiko des Kunden. Diese Funktion wird ebenfalls nicht vom OMS selbst durchgeführt und stellt, wie auch die Anfrage einer Quote, einen Serviceaufruf an ein anderes System dar. Ist der Kunde kreditwürdig, so wird mit dem Verlauf des Prozesses fortgefahren. Wenn nicht, endet der Prozess frühzeitig und der Kunde wird über das Ereignis informiert. Konkret bedeutet das, dass in diesem Fall kein Geschäft abgeschlossen werden konnte.

Sind alle Prüfungen erfolgreich, so wird ein Geschäft mit dem Status „Pending“ erstellt. Anschließend wird ein an dem Prozess beteiligter Benutzer seitens der Bank damit beauftragt, Prozessrelevante Daten an das OMS zu übermitteln. Beispielsweise könnte es sein, dass kundenrelevante Daten, sowie Daten für die Abwicklung der Zahlung, auf manuellem Wege an den Prozess übergeben werden müssen. Die manuelle Interaktion eines Benutzers ist dann gefragt, wenn die Aktivität entweder noch nicht automatisiert wurde oder nicht automatisiert werden kann. An diesem Punkt wartet der FX-Spot Order Prozess bis die Dateneingabe abgeschlossen ist, wobei die Eingabe durch den Benutzer über eine sogenannte Arbeitsliste (Worklist) durchgeführt wird (siehe 2). In Bezug auf den Life-Cycle einer Order (nicht des Prozesses) bedeutet das Warten auf weitere Informationen, dass der Zustand der Order auf „Pending“ gesetzt wird. Nachdem der Benutzer die benötigten Daten an das OMS übermittelt hat, sind alle zur Erstellung der Order benötigten Prozessschritte abgearbeitet. Nun kann der Zustand der Order auf „Active“ gesetzt werden. Im Falle einer FX-Spot Order, werden ab diesem Zeitpunkt in der Regel zwei Geschäftstage gewartet bis der eigentliche Geldtransfer vollzogen wird. Während dieser Zeit hat der Kunde die Möglichkeit, die Order entweder zu stornieren oder zu verändern. Im Falle einer Stornierung wird die Verarbeitung der Order ordnungsgemäß gestoppt und der Orderstatus auf „Cancled“ gesetzt. Für den Fall, dass der Kunde die Order verändern möchte, wird die Verarbeitung der Order ebenfalls gestoppt und der Orderstatus auf „Replaced“ gesetzt. Daraufhin wird eine neue Order, abhängig der Änderungen erteilt und ein neuer Prozess gestartet. Nimmt der Kunde weder eine Stornierung noch eine Änderung vor, so wird nach Ablauf der zwei Geschäftstage der eigentliche Geldtransfer vom OMS eingeleitet. Das bedeutet, dass das OMS den Zahlungsauftrag an das dafür zuständige Zahlungssystem übergibt. Nachdem das OMS vom Zahlungssystem über den Erfolg der Zahlungsabwicklung benachrichtigt wurde, wird der Status der Order vom OMS auf „Matured“ gesetzt. Die Order ist somit abgeschlossen.

4.3.2 Technische Sicht

Der aus fachlicher Sicht beschriebene FX-Spot Order Prozess, soll nun unter Verwendung von BPMN 2.0 modelliert werden. Dabei sind die Funktionen der einzelnen Aktivitäten in der

fachlichen Sicht beschrieben und werden daher nur noch namentlich erwähnt. Auch wird für die folgende Beschreibung ein Verständnis der grundlegenden BPMN 2.0 Prozesselemente vorausgesetzt (BPMN Offensive Berlin, 2011).

Abbildung 3 zeigt den FX-Spot Order Prozess, modelliert in BPMN 2.0. Nach dem Start-Event trifft der Prozess auf einen Service-Task, welcher die vom Kunden übermittelte Quote validiert. Da das OMS die Gültigkeit der vom Kunden übermittelten Quote bereits kennt, stellt die Validierung im Wesentlichen nur einen Vergleich zweier Werte dar. Aus diesem Grund kann diese Prüfung mit Hilfe einer internen Funktion des OMSs durchgeführt werden. Anschließend wird das Ergebnis der Validierung mit Hilfe eines exklusiven Gateways überprüft. War die Validierung fehlerhaft, so wird der Prozessverlauf unmittelbar auf ein End-Event geleitet. Im Falle einer erfolgreichen Validierung, wird anschließend das Kreditrisiko des Kunden mit Hilfe eines weiteren Service-Tasks geprüft. Dieser Service-Task ruft dann einen Service eines externen Systems auf. Der Prozess verläuft an dieser Stelle synchron, wobei auf die Antwort des jeweiligen Systems gewartet wird. Nachdem eine Antwort erhalten wurde, wird der Prozess bei einem Fehler, durch ein exklusives Gateway, auf ein End-Event geleitet. War die Prüfung erfolgreich, so trifft der Prozess auf einen User-Task. Dieser Task erfordert die manuelle Eingabe eines an dem Prozess beteiligten Benutzers. An dieser Stelle muss, laut der fachlichen Prozessbeschreibung, der Status der Order auf „Pending“ gesetzt werden. Zustandsänderungen sind jedoch bewusst nicht modelliert und werden in 4.6 erläutert. Nach erfolgreicher Bearbeitung des User-Tasks, trifft der Prozess auf ein Event-Gateway. Abhängig davon, welches der drei auf das Event-Gateway folgenden Events als erstes ausgelöst wird, wird entschieden welcher Prozesspfad weiter durchlaufen wird. Das „Timer Catching Event - Mature“, stellt eine Stoppuhr dar, welche bis zum Eintreten eines bestimmten Zeitpunkts wartet und dann ein Event auslöst. Bezogen auf die fachliche Sicht bedeutet dies, dass dies Stoppuhr nach zwei Geschäftstagen das Event auslöst und somit den eigentlichen Geldtransfer initiiert. Auf das Event folgen in diesem Beispiel dann weitere Service-Tasks, welche zur Abwicklung ausgeführt werden müssen. Die beiden Events „Signal Catching Event - Delete“ und „Signal Catching Event - Update“ achten auf das Eintreten von Events, die durch den Kunden ausgelöst werden können. Möchte dieser die Order stornieren oder verändern, so ruft er den dafür zuständigen, vom OMS angebotenen, CRUD-Service auf. Dadurch wird das Auslösen des jeweiligen Events eingeleitet. Anschließend werden dann, je nach ausgelöstem Event, die nötigen Operationen (Tasks) bearbeitet. Darauf wird jedoch in diesem Beispiel nicht weiter eingegangen. Abschließend besitzt jeder einzelne Prozesspfad, welcher auf eines der drei erläuterten Events folgt, ein eigenes End-Event.

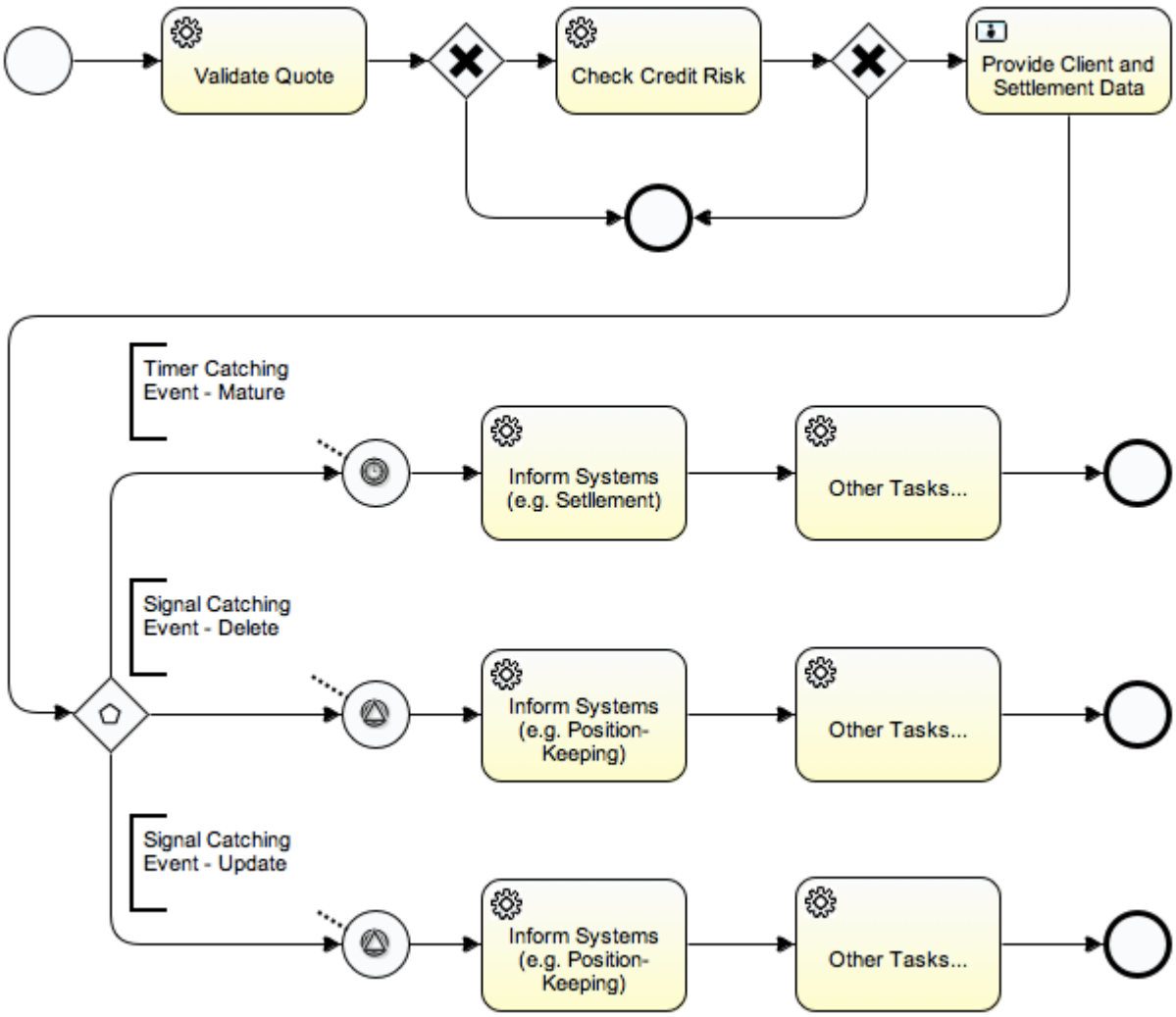


Abbildung 3: FX-Spot Order Beispielprozess

4.4 Modularität

Eine Anforderung an das Order-Management-System ist die Modularität von Prozessen (siehe 3.1). Modularität bedeutet in diesem Zusammenhang, dass ein zu designender Gesamtprozess, aus einzelnen verschiedenen „Modulen“ (Subprozesse) zusammensetzt werden kann. Jedes Modul führt dabei logisch zusammenhängende Arbeitsschritte durch, die einen Teil des gesamten Prozesses darstellen. Die Zerlegung eines Prozesses in einzelne Module hat den Vorteil, dass es unterschiedliche Versionen einzelner Module geben kann und diese dann zu einem vollständigen Geschäftsprozess miteinander kombiniert werden können. Beispielsweise könnten sich zwei Orders, beziehungsweise Order-Typen, dahingehend unterscheiden, dass diese bei einer Änderung (Update), unterschiedliche Businesslogik verwenden. Um beide Prozesse lauffähig machen zu können, müssen nun lediglich unterschiedliche Update-Module implementiert werden, wobei der restliche Teil des Prozesses gleich bleibt. Auch können die einzelnen Module eines Prozesses unabhängig voneinander implementiert

werden. Ein Entwickler kann sich beispielsweise mit der Implementierung eines Update-Moduls beschäftigen, während ein anderer an einem Delete-Modul arbeitet. Dies bietet erhebliche Vorteile bei der Implementierung neuer Geschäftsprozesse hinsichtlich der Kosten und der Zeit. Gleichermaßen wird die Wartbarkeit erhöht, da bei Änderungen nur ein Teil des Prozesses betrachtet werden muss.

4.4.1 Modularität des Life-Cycles

Um Geschäftsprozesse in modulare Einheiten einteilen zu können, wird der vorgegebene Life-Cycle aus 2.1 zur Hilfe genommen. Wie bereits erwähnt wurde dieser in Bezug auf dieses Thesis-Projekt vorgegeben und soll in erster Linie zur Erarbeitung des standardisierten Entwurfskonzepts des OMS dienen. Der Order Life-Cycle besteht aus insgesamt fünf Zuständen. Wurde eine Order an das OMS übergeben, so durchläuft diese während der Erstellung die beiden Zustände „Pending“ und „Active“. Erst nachdem der Status „Active“ erreicht wurde, können weitere Zustände folgen. Daher können die beiden erwähnten Zustände als Bestandteil des Erstellungs-Prozesses einer Order betrachtet werden. In Bezug auf die Modularität, wird dieser Erstellungs-Prozess, einer modularen Einheit zugewiesen und im weiteren Verlauf als Create-Modul bezeichnet. Die restlichen Zustände „Matured“, „Deleted“ und „Updated“ können als atomar angesehen werden, da zur Erreichung des Zustands jeweils logisch getrennte Prozessabschnitte durchlaufen werden (Transaktionsklammer). Daraus ergeben sich mit Hilfe des Order-Life-Cycles insgesamt die vier Module Create, Mature, Delete und Update.

4.4.2 Modularität des Beispielprozesses

Unter Verwendung der in 4.4.1 beschriebenen Module, soll nun auch der FX-Spot Order Beispielprozess modularisiert werden. Dazu ist es notwendig, den Beispielprozess so zu unterteilen, dass die einzelnen Prozessabschnitte der Erreichung eines Zustands im Order Life-Cycle dienen. Jeder Teilprozess stellt dann einen Subprozess dar, welcher jeweils separat modelliert ist. Am Ende jedes einzelnen Moduls bzw. eines Subprozesses, findet dann ein Zustandswechsel, abhängig vom verwendeten Life-Cycle-Modell, statt. Beispielsweise findet am Ende des Create-Moduls, ein Zustandswechsel der Order auf den Status „Active“ statt.

Life-Cycle Änderungen werden wie bereits erwähnt in einem Objekt gespeichert, welches für jede Instanz eines Prozesses und jedes Prozess-Modell eindeutig ist (siehe 4.1.3). Im Falle eines modularen Prozesses, stehen jedem einzelnen Modul des Prozesses ein eigenes (Daten)-Objekt zur Verfügung. Aus Sicht eines modularen Prozesses mit mehreren Subprozessen bedeutet dies, dass das jeweilige Objekt beim Starten eines Subprozesses als Eingabe-

parameter übergeben wird und nach erfolgreicher Beendigung, als Rückgabeparameter an den modularen Prozess zurückgegeben wird. Dies wird in Kapitel 4.9.3.3 näher erläutert.

Abbildung 4 zeigt den Beispielprozess aus Abbildung 3 unterteilt in die vier, zuvor erläuterten Module Create, Mature, Delete und Update. Der restliche, nicht eingefärbte Prozessverlauf, kann jedoch keinem Modul zugewiesen werden, da dieser für den grundlegenden Verlauf des Prozesses entscheidend ist. Auch stellt der Prozessverlauf einen Teil der Verhaltensorientierten Betrachtungsweise dar (siehe 2). Die Vorgehensweise zur Implementierung einzelner Module, ist Kapitel 4.9 erklärt.

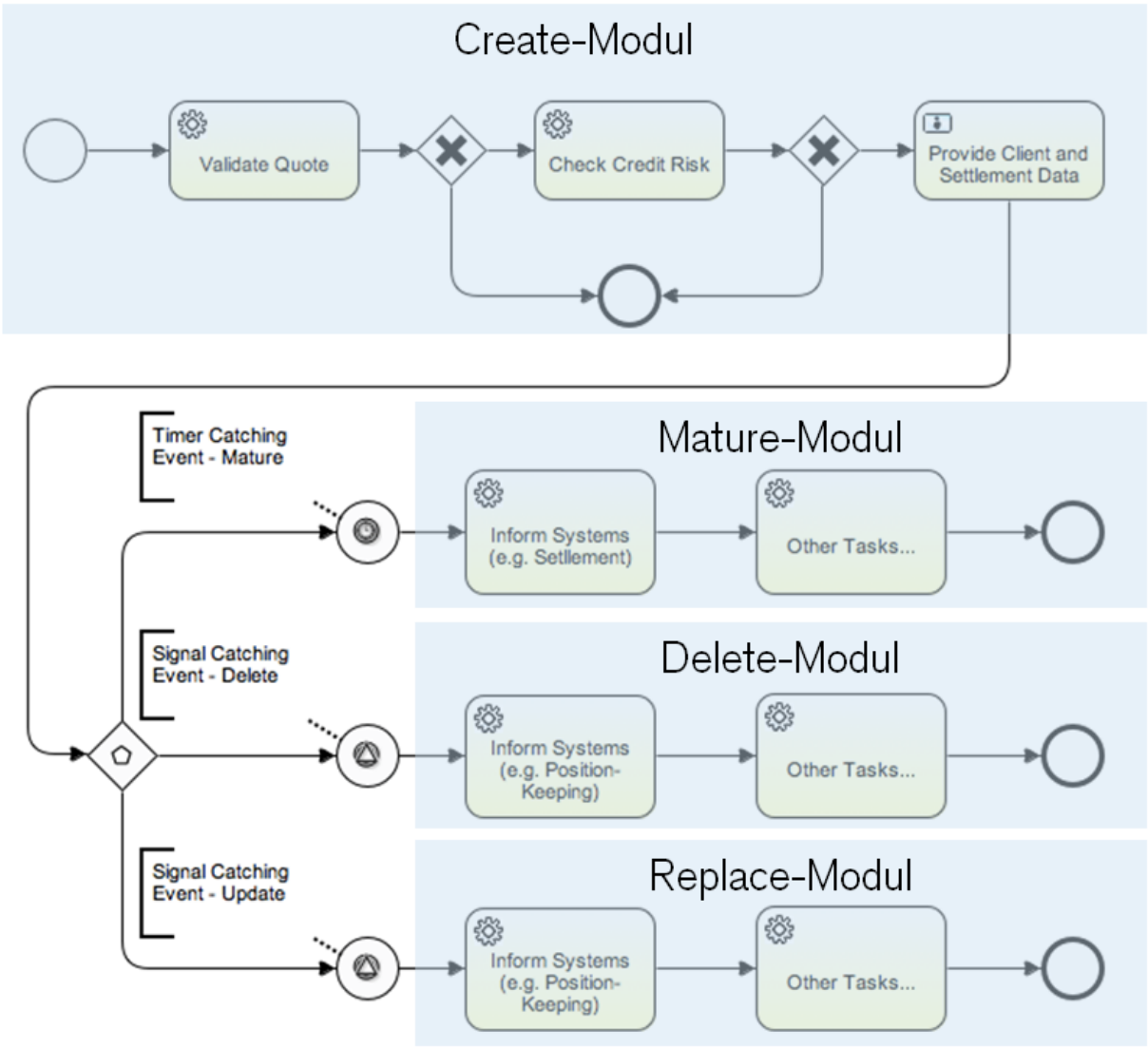


Abbildung 4: FX-Spot Order Beispielprozess - Modulare Einheiten

4.5 Das generische Prozess-Template

Abbildung 5 zeigt das generische Prozess-Template, dass mit Hilfe von Abbildung 4 entstanden ist. Das Template enthält, genau wie der Beispielprozess, ein Event-Gateway mit drei ausgehenden Prozesspfaden. Jeder dieser Prozesspfade kann von einem eintreffenden Event ausgelöst werden. Lediglich die einzelnen Aktivitäten vor und nach dem Gateway und den Events, wurden innerhalb der erläuterten vier Module zusammengefasst. Aus Sicht von BPMN 2.0, stellt jedes dieser Module einen Subprozess dar, was durch ein „+“ unterhalb des Namens gekennzeichnet ist. Dabei wird zwischen verschiedenen Arten von Subprozessen unterschieden. Im Folgenden werden sogenannte „Call Activities“ verwendet, welche einen wiederverwendbaren Subprozess darstellen. Jeder dieser Call Activities wird in einem separaten Prozessdiagramm modelliert und kann dann von dem Elternprozess, also dem Prozess-Template, aufgerufen werden. Auch können jedem Subprozess Parameter übergeben werden und von diesem, auch wieder an den Elternprozess zurückgegeben werden.

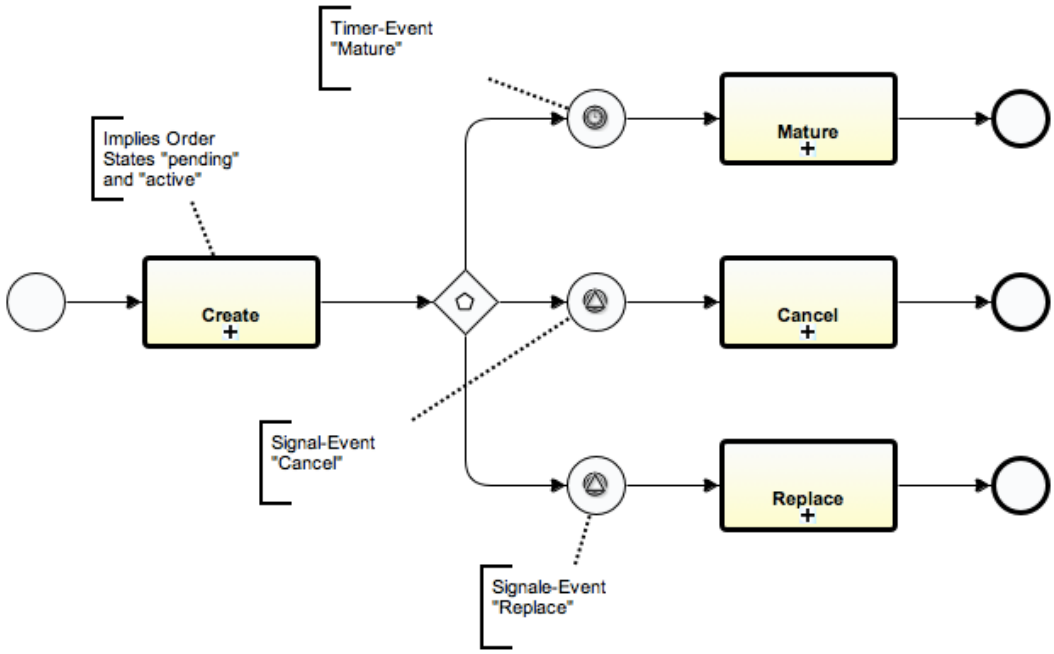


Abbildung 5: Generisches Prozess-Template

Legt man die modellierten Prozesse zweier verschiedener Order-Typen nebeneinander, so erkennt man, dass diese unterschiedliche Prozesse abbilden. Beispielsweise könnten sich die beiden Orders dahingehend unterscheiden, dass diese unterschiedliche Logik beim Erstellen einer Order implementieren. Das würde zur Folge haben, dass es für jeden dieser Prozesse ein eigenes Create-Modul gibt, wobei die restlichen Module wiederverwendet werden können. Die verwendeten Module, bzw. Subprozesse, können dann beim Starten einer Instanz des generischen Prozess-Templates, abhängig vom gewünschten Order-Typ, konfi-

guriert werden. Die genaue Erklärung dieser Funktionsweise, wird in Kapitel 4.9.3.2 näher beschrieben.

Anhand des Templates lässt sich sehr gut erkennen, wie Life-Cycle einer Order und der jeweilige Geschäftsprozess miteinander korrespondieren. Die Module des Templates, bilden bis auf den Status „Pending“ und „Active“, alle Zustände des Life-Cycle ab. Die beiden Zustände sind dabei innerhalb des Create-Moduls zusammengefasst. Prozesse die sich nicht mit Hilfe des Prozess-Templates modellieren lassen, können ein alternatives Template verwenden. In diesem Fall muss das Template die gegebenen Module Create, Mature, Cancel und Replace enthalten, wobei der eigentliche Prozessverlauf in Bezug auf die Verhaltensorientierte Betrachtungsweise variieren kann. Bei der Verwendung eines alternativen Life-Cycles, sind zudem andere Module denkbar.

4.6 Modellierung von Statusänderungen einer Order

Wie bereits erwähnt, muss zwischen dem Life-Cycle einer Prozess-Instanz und dem Life-Cycle einer Order unterschieden werden. Der Life-Cycle einer Prozess-Instanz wird von der Prozess-Engine bzw. der BPM Plattform selbst verwaltet und ist nicht unter Kontrolle des OMS. Der Life-Cycle einer Order hingegen, muss vom OMS verwaltet werden. Es ist vorgesehen, dass aus implementationstechnischer Sicht, Zustandsänderungen im Order-Life-Cycle, abhängig von Übergabeparametern über die Selbe Klasse bzw. Methode abgehandelt werden. Dies wird in Kapitel 4.9.3.4 beschrieben. In den folgenden Abschnitten wird gezeigt, wie Zustandsänderungen einer Order in Bezug auf die Prozess-Modellierung organisiert sind.

4.6.1 Modellierung im generischen Prozess-Template

In Activiti werden während der Ausführung von Prozess-Instanzen, an unterschiedlichen Stellen Events ausgelöst. Beispiele hierfür sind das Starten und Beenden von Service-Tasks, Subprozessen oder Gateways. Um aufgrund des Eintretens eines Events, vom OMS gewisse Operationen durchführen zu lassen, stellt Activiti sogenannte „Execution-Listener" zur Verfügung. Diese werden als Bestandteil eines einzelnen Prozess-Elements, wie z.B. eines Subprozesses modelliert.

In Bezug auf das in 4.5 erläuterte generische Prozess-Template, stellt das Create-Modul den atomaren Teilprozess einer Order dar, welcher zur Erzeugung der Order durchlaufen werden muss. Konkret betrifft dies den Teil des Gesamtprozesses, bei dem am Ende der Status der Order auf „Active“ gesetzt werden kann. Ein Beispiel für ein Create-Modul in Bezug auf den FX-Spot Order Beispielprozess, wird in 4.9.2 vorgestellt. Beim Abschließen des Create-Moduls, soll nun ein eintretendes Event durch einen Execution-Listener abgefangen werden.

Dieser leitet dann die Statusänderung der Order auf den Status „Active“ ein. Gleiches Vorgehen gilt auch für die restlichen drei Module des Templates, wobei der Status jeweils auf „Matured“, „Cancled“ oder „Replaced“ gesetzt wird. Der Vorteil dieser Art der Prozessmodellierung ist, dass man in Bezug auf ein generisches, modulares Prozess-Template die Statusänderungen einer Order fest in das Template integrieren kann. In Kapitel 4.9.3.4 wird gezeigt, wie dies innerhalb des Prototyps implementiert ist.

Mit der Modellierung von Statusänderungen einer Order im generischen Prozess-Template, sind alle Zustände des Order-Life-Cycles bis auf „Pending“ abgedeckt. Da dieser nicht obligatorisch ist, kann es auch Order-Typen geben, die diesen Status nicht annehmen. Ist diese Zustandsänderung dennoch notwendig, kann dies unter Verwendung der Methode in 4.6.2 erfolgen.

4.6.2 Spezialfall: Modellierung des Status „Pending“

Der Orderstatus „Pending“, ist nicht für jeden Order-Typ obligatorisch. Im Create-Modul des FX-Spot Order Beispielprozesses aus Abbildung 9 muss diese Statusänderung jedoch während der Verarbeitung des erfolgen. Dabei soll vor der Durchführung des User-Tasks „Manuelle Bonitätsprüfung“ die Statusänderung erfolgen. Vor dem eigentlichen Start der Aktivität kann auch hier ein von der Prozess-Engine geworfenes Event abgefangen werden. Dazu wird ein Execution-Listener innerhalb des User-Tasks integriert, welcher dann die Statusänderung einleitet. Bei der Modellierung eines Create-Moduls muss demnach entschieden werden, ob der Status „Pending“ angenommen werden soll und wann in diesen gewechselt werden soll.

4.6.3 Synchronisierung von Order- und Prozess-Life-Cycle

Wie bereits erwähnt, verwaltet das OMS den Order-Life-Cycle, während die Prozess-Engine den Life-Cycle des Prozesses kontrolliert. Auf den Prozess-Life-Cycle hat das OMS daher selbst keinen Einfluss. Dennoch muss sichergestellt sein, dass beide Life-Cycle miteinander synchronisiert werden. Im Folgenden wird erläutert, wie das OMS diese Thematik behandelt.

Die Statusänderungen einer Order sind innerhalb des generischen Prozess-Templates fest integriert. Dabei wird nach Durchführung eines jeden Moduls (Subprozesses), aus Sicht des OMS, der jeweilige Statuswechsel der Order vollzogen. Aus Sicht der Prozess-Engine, besitzt eine Prozess-Instanz nach Abschluss eines Moduls (Subprozesses) ebenfalls einen bestimmten Fortschritt im Prozessverlauf. Demnach bedeutet ein Wechsel des Orderstatus aus Sicht des OMS, gleichzeitig ein Wechsel des Prozessstaus aus Sicht der Prozess-Engine. Auf diese Weise kann das OMS nachvollziehen, an welchem Punkt sich eine jede Prozess-Instanz in Bezug auf das generische Prozess-Template befindet.

4.7 Technische Standardarchitektur

In diesem Kapitel wird die technische Standardarchitektur des OMS nach einer Methode zur Architekturbeschreibung mit dem Namen Quasar beschrieben (Siedersleben, 2003). Dabei werden zuerst die grundlegenden Anwendungskomponenten beschrieben. Danach wird die externe Sicht des OMS erläutert, wobei auf die angebotenen und konsumierten Services eingegangen wird. Abschließend wird die interne Sicht des OMS beschrieben. Dabei wird auch das Verhalten der dargestellten Anwendungskomponenten untereinander erläutert. Das Ergebnis dieses Kapitels ist dann die technische Standardarchitektur, welche für alle Implementationen des OMS als Entwurfskonzept (Blueprint) dienen soll.

4.7.1 T-Komponenten

Nach Quasar, werden Komponenten der technischen Standardarchitektur als sogenannte „T-Komponenten“ bezeichnet. T-Komponenten zeichnen sich dadurch aus, dass die jeweilige Funktionsweise für Implementation der Standardarchitektur gleichbleiben ist. Im Folgenden werden die T-Komponenten des OMS erläutert. Später werden diese dann zur Beschreibung der externen, sowie der internen Sicht des OMS verwendet.

Service-Manager: Der Service-Manager, kurz *SM*, dient als eine Art Fassade für das OMS und bietet externen Systemen die Möglichkeit neue Orders aufzugeben und diese während der Verarbeitung zu verwalten. Dafür werden den externen Systemen, SOA-Services zur Verfügung gestellt, wobei externe Systeme sowohl User- als auch Non-User-Systeme sein können. Die Services werden dabei als CRUD-Services bezeichnet (siehe 4.1.1). Dabei spiel es keine Rolle mit Hilfe welcher Technologie die Services implementiert sind. Diese könnten als REST-Services, Web-Services, JMS-Services etc., umgesetzt werden. Externe Systeme, welche mit dem OMS kommunizieren wollen, werden dazu eine Anfrage mit den nötigen Informationen an den jeweiligen CRUD-Service des SM senden. Dieser nimmt die Anfrage entgegen, wertet diese aus und leitet die notwendigen Schritte zur Verarbeitung der Anfrage ein. Möchte ein externes System beispielsweise eine FX-Spot Order über das OMS initiieren, so stellt dieses eine Anfrage an den Create-Service des SM. Der SM übergibt die Anfrage dann an den Request-Storage (siehe unten). In dem Request-Storage wird die Anfrage persistiert und dann für die Verarbeitung durch das OMS zur Verfügung gestellt. Eine weitere Funktion des SMs ist, dass dieser auch Anfragen welche vom OMS selbst stammen, aus dem Request-Storage lesen kann. Eine laufende Prozess-Instanz einer FX-Spot Order besitzt beispielsweise einen Service-Task, welcher Informationen von einem externen System anfordern soll (siehe Abbildung 3, „Check Credit Risk“). Diese Anfrage wird von der dafür zuständigen Komponente des OMS in den Request-Storage geschrieben und steht dann auch dem SM als ausgehende Anfrage zur Verfügung. Der SM schreibt also eingehende

Anfragen externer Systeme in den Request-Storage. Gleichermaßen werden ausgehende Anfragen des OMS aus dem Request-Storage gelesen und an die entsprechenden Systeme weitergeleitet. Die Antworten der ausgehenden Anfragen werden von dem SM ebenfalls in den Request-Storage geschrieben, wo diese dann der jeweiligen Komponente bzw. Prozess-Instanz des OMS zur Verfügung stehen.

Order-Manager-Engine: Die Order-Manager-Engine *(OME)*, ist die Hauptkomponente des OMS. Der Kern besteht im Wesentlichen aus zwei Teilen, der BPM Plattform und der Business-Logik. In Bezug auf die BPM Plattform kann eine beliebige Open-Source oder auch kommerzielle Lösung für die die Modellierung, Ausführung und Steuerung von Geschäftsprozessen verwendet werden. Jedoch ist es an dieser Stelle notwendig, eine Plattform zu verwenden, deren Prozess-Engine sich in das OMS integrieren lässt. Dazu muss diese ein API zu Verfügung stellen, welches unter Verwendung der Zielsprache des OMS verwendet werden kann. Welche Plattform zum Einsatz kommen soll, hängt dabei in erster Linie vom jeweiligen Funktionsanspruch des OMS ab.

Die Business-Logik enthält die Implementationen der einzelnen Prozesse. Beispielsweise ist die Logik einer einzelnen Aktivität, welche innerhalb eines Prozesses modelliert wurde, als Business-Logik implementiert. Bei der Modellierung eines Prozesses, kann diese dann der Aktivität zugewiesen werden. Eine Prozess-Instanz greift jedoch nicht direkt auf die ihr zugewiesene Business-Logik zu, sondern ruft diese nur indirekt auf. Somit sollen Prozess-Engine und Business-Logik voneinander unabhängig sein. Ein Beispiel für eine mögliche Umsetzung dieses Vorgehens, ist in Kapitel 4.9.3.6 beschrieben.

Die OME hat zudem die Aufgabe, Anfragen welche vom Service-Manager in den Request-Storage (siehe unten) geschrieben wurden, zu lesen und auszuführen. Auch ist diese Komponente dafür zuständig, Anfragen von einzelnen Prozess-Instanzen an Services externer Systeme in den Request-Storage zu schreiben. Ab diesem Zeitpunkt muss die OME auf die jeweilige Antwort warten, welche ebenfalls aus dem Request-Storage gelesen wird und dann der anfragenden Prozess-Instanz zurückgeliefert wird.

Request-Storage: Der Request-Storage *(kurz RS)* stellt eine Art interne Queue dar, welche den Service-Manager (SM) und die Order-Manager-Engine (OME) voneinander entkoppeln soll. Die ein- und ausgehenden Anfragen des OMS werden dabei in der Queue persistiert. Eingehende Anfragen stammen dabei von externen Systemen, welche vom SM in den RS geschrieben werden. Ausgehende Anfragen stammen hingegen vom OMS selbst, wobei diese von einzelnen Prozess-Instanzen gestellt und von der Order-Manager-Engine in den RS geschrieben werden. Nachdem ausgehende Anfragen vom SM gelesen und an das ent-

sprechende System weitergeleitet wurden, werden die Ergebnisse der Anfragen ebenfalls im RS zur Verfügung gestellt. Diese können dann von der OME gelesen werden und an die entsprechende Prozess-Instanz weitergeleitet werden, welche die Anfrage gestellt hat. Nach einem Systemabsturz, sollen alle Informationen welche in den RS geschrieben wurden, nach wie vor zur Verfügung stehen. Daher ist es notwendig, dass der RS die Anfragen und Ergebnisse persistieren kann. An dieser Stelle ist zu erwähnen, dass die gespeicherten Daten des RS nicht komplex sind, sondern lediglich Anfragen an bestimmte Funktionalitäten des OMS darstellen.

Quote-Engine: Die Quote-Engine *(QE)*, liefert die richtigen Quotes (Preise) für die einzelnen, angebotenen Produkte bzw. Dienstleistungen des OMS. Ein Client, kann beispielsweise eine Anfrage für den aktuellen Kurs eines gewünschten Währungspaars an das OMS senden. Das OMS ruft dann mit Hilfe des Service-Managers die dafür zuständigen, externen Systeme auf und liefert die Information an den Client zurück. Die QE orchestriert also externe Services und stellt die Funktionalitäten den Clients des OMS zur Verfügung. In dieser Arbeit wird jedoch auf diese Komponente aus zeitlichen Gründen nicht weiter eingegangen.

Data-Access: Die Data-Access *(DA)* Komponente regelt den Lese- und Schreibzugriff zwischen der OME und der OMS-Database. Im Falle einer relationalen Datenbank regelt diese Komponente auch das Objekt-Relationale Mapping der (Daten)-Objekte.

OMS-Database: In der OMS-Database *(OMS-DB)* werden alle notwendigen Daten einer Order gespeichert. Zu speichernde Daten in Bezug auf eine FX-Spot Order (siehe Abbildung 3) sind z.B. das gewünschte Währungspaar, der zu kaufende/verkaufende Betrag, Käufer- und Verkäufer bezogene Daten, etc. Prozess-Instanz bezogene Daten, sind Daten, welche zur Verwaltung und Steuerung eines Prozesses verwendet werden. Ein Beispiel hierfür ist der aktuelle Stand in dem Life-Cycle eines Prozesses. Diese Daten stehen nicht unter Kontrolle des OMS, sondern werden von der verwendeten BPM Plattform bzw. der Prozess-Engine selbst verwaltet. Demnach verwendet die Prozess-Engine auch ein eigenes Datenbankschema. Folglich werden in der OMS-DB alle Order-bezogenen Daten gespeichert, wobei Prozess-bezogene Daten von der Prozess-Engine selbst persistiert werden.

4.7.2 Beschreibung der externen Sicht

In diesem Abschnitt wird die externe Sicht des OMS beschrieben, wobei zwischen angebotenen und konsumierten Services unterschieden wird. Somit wird festgelegt wie externe Systeme, Funktionalitäten des OMS verwenden können. Gleichermaßen wird definiert, wie das OMS selbst die Funktionalitäten externer Systeme verwendet.

4.7.2.1 Angebotene Services

Clients, stehen zur Kommunikation mit dem OMS diverse Schnittstellen zur Verfügung. Wie in 3.1 beschrieben, soll das System dazu CRUD-Services anbieten. Zusätzlich soll das OMS Services für das Anfragen von Preisen (Quotes) bestimmter Produkte bzw. Dienstleistungen anbieten. Im Folgenden werden diese Schnittstellen unter Verwendung von Java definiert. Die Services sind dann innerhalb des Service-Managers zu implementieren und über einen Web-Applikations-Server (kurz *WAS*) zur Verfügung zu stellen. Zur Beschreibung der Schnittstellen, werden die Methoden-Signaturen der einzelnen Services kurz erläutert und mit Hilfe von Java beschrieben.

Create-Service: Der Create-Service bietet externen Systemen die Möglichkeit neue Orders zu erteilen bzw. zu erstellen. Beim Aufruf des Services werden dazu Order-spezifische Parameter übergeben. Anhand dieser Parameter, kann das OMS beispielsweise erkennen um welchen Order-Typ (z.B. FX-Spot Order) es sich handelt. Dadurch kann dann der Start einer neuen Prozess-instanz des generischen Prozess-Templates eingeleitet werden. Der Order-Typ bestimmt dabei welche Subprozesse für jedes Modul des generischen Prozess-Templates ausgeführt werden sollen (siehe 4.5). Nach dem Aufruf des Services erhält der Client vom OMS eine Antwort über Erfolg oder Misserfolg.

```
public CreateResponse createOrder(Map<String, Object> variables);
```

Update-Service: Der Update-Service bietet externen Systemen die Möglichkeit, eine aktive Order zu verändern. Eine Änderung könnte z.B. einen einfachen skalaren Wert betreffen oder aber auch den gesamten Order-Typ. Wird dieser Service aufgerufen, wird die laufende Prozess-Instanz beendet und anschließen eine neue Order zusammen mit den geänderten Daten erteilt. Nach dem Aufruf des Services, erhält der Client vom OMS eine Antwort über Erfolg oder Misserfolg.

```
public UpdateResponse updateOrder(long orderId, Map<String, Object>
variables);
```

Delete-Service: Der Delete-Service bietet externen Systemen die Möglichkeit, eine aktive Order zu stornieren. Nach dem Aufruf des Services, erhält der Client vom OMS eine Antwort über Erfolg oder Misserfolg.

```
public DeleteResponse deleteOrder(long orderId);
```

Read-Service: Der Read-Service bietet externen Systemen die Möglichkeit, eine Order abhängig von einer eindeutigen ID zu suchen. Nach dem Aufruf des Services erhält der Client vom OMS eine Antwort mit den Ergebnissen der Anfrage. Im Falle des Read-Services besteht die Antwort jedoch maximal aus einem Ergebnis.

```
public ReadResponse getOrder(long orderId);
```

Search-Service: Der Search-Service stellt eine Verfeinerung des Read-Services dar. Der Unterschied besteht darin, dass man nicht nur nach einer Order mit einer eindeutigen ID suchen kann, sondern auch Suchparameter übergeben kann, um die Suche zu verfeinern. Anstelle der Rückgabe eines einzigen Ergebnisses, können hier mehrere Ergebnisse zurückgegeben werden.

```
public List<Object> getOrders(Map<String, Object> variables);
```

Request-For-Quote-Service: Der Request-For-Quote-Service bietet einem Client des OMS die Möglichkeit, einzelne Preise (Quotes) für verschiedene Produkte bzw. Dienstleistungen anzufordern. Wie in 4.3.1 beschrieben, könnte ein Client beispielsweise den Preis, d.h. den aktuellen Kurs, für ein bestimmtes Währungspaar anfragen. Wichtig ist, dass das OMS diesen Service nicht selbst implementiert. Dieser wird lediglich konsumiert und durch das OMS den jeweiligen Clients zur Verfügung gestellt.

```
public BigDecimal getRequestForQuote(Map<String, Object> variables);
```

Streamed-Quote-Service: Der Streamed-Quote-Service stellt eine Erweiterung des Request-For-Quote-Services dar. Anstatt dem anfragenden Client nur einen einzigen Preis (Quote) zurückzuliefern, können mit Hilfe dieses Services mehrere Preise (Quotes) über einen bestimmten Zeitraum zurückgegeben werden.

```
public ObjectOutputStream getStreamedQuote(Map<String, Object>
variables);
```

4.7.2.2 Konsumierte Services

Im Folgenden wird beschrieben, welche unterschiedlichen Services durch das OMS konsumiert werden. Dabei wird im Wesentlichen zwischen Quote-Services, Worklist-Services und Funktionale-Services unterschieden.

Quote-Services: Clients des OMS haben die Möglichkeit, Preise (Quotes) für verschiedene Produkte bzw. Dienstleistungen anzufragen. Die Preise (Quotes) werden dabei jedoch nicht vom OMS selbst bereitgestellt. Um diese Funktionen anzubieten, orchestriert das OMS lediglich die Services externer Systeme wie z.B. Preis-Engines und stellt die Funktionalitäten den Clients zur Verfügung. Das Aufrufen dieser externen Services wird jedoch innerhalb dieser Arbeit nicht weiter behandelt.

Worklist-Services: Bei der Ausführung von Prozess-Instanzen durch die Prozess-Engine kann es vorkommen, dass zur Verarbeitung einzelner Aktivitäten die Interaktion eines menschlichen Benutzers benötigt wird. Dazu muss dem jeweiligen Benutzer ein System zur Verfügung stehen, welches es ihm ermöglicht die ihm zugeteilten Aufgaben zu bearbeiten und abzuschließen. Dieses System wird im Allgemeinen als Worklist bezeichnet. Das OMS bietet jedoch selbst keine Worklist an und verwendet daher eine zentrale Lösung. Wird die manuelle Bearbeitung einer Aktivität durch einen Benutzer gefordert, so ruft das OMS einen Service der zentralen Worklist auf. Nach der Bearbeitung der Aktivität durch den entsprechenden Benutzer, liefert die Worklist dann die nötigen Informationen an das OMS zurück. Da die Verwendung einer Worklist kein Bestandteil dieser Arbeit ist, wird auf diese konsumierten Services nicht weiter eingegangen.

Funktionale-Services: Sind konsumierte Services externer Systeme, die für die Verarbeitung von einzelnen Prozessaktivitäten verwendet werden. Die Verwendung eines solchen Services ist für alle Funktionalitäten notwendig, die nicht vom OMS selbst implementiert und als interne Funktion zur Verfügung gestellt werden.

4.7.3 Beschreibung der internen Sicht

In diesem Abschnitt wird die innere Sicht der technischen Standardarchitektur des OMS erklärt. Im ersten Schritt wird dazu das gegenseitige Verhalten der in 4.7.1 vorgestellten T-Komponenten erläutert und grafisch dargestellt. Anschließend werden die T-Komponenten durch einige Subkomponenten verfeinert, welche ebenfalls Teil der technischen Standardarchitektur sind.

Abbildung 6 zeigt die zuvor erläuterten T-komponenten des OMS. Die ausgehenden Pfeile der Komponenten stellen den Gegenseiten Zugriff der Komponenten untereinander dar. Zur

Verteilung der einzelnen Komponenten lässt sich sagen, dass Service-Manager (SM), Request-Storage (RS), OMS-Database (OMS-DB) und Order-Manager-Engine (OME) einerseits zusammen auf einer Maschine installiert werden können. Andererseits können diese jedoch auch getrennt voneinander, auf separaten Maschinen verteilt werden. Es wird empfohlen, Data-Access (DA) und OME zusammen auf einer Maschine zu installieren. Diese sind zwar logisch voneinander getrennt, jedoch würde eine physische Trennung an dieser Stelle keinen Sinn machen.

Auf der einen Seite schreibt der SM eingehende Service-Anfragen von Client-Systemen in den RS, welche wiederrum von der OME aus dem RS gelesen und die nötigen Operationen durchgeführt werden. Auf der anderen Seite schreibt die OME Service-Anfragen von einzelnen Prozess-Instanzen in den RS. Diese werden von dort durch den SM gelesen, welcher anschließend den jeweiligen Service eines externen Systems aufruft. Alle eine Order betreffende Daten, werden durch die OME mit Hilfe der DA-Komponente in die OMS-DB geschrieben bzw. aus der OMS-DB gelesen.

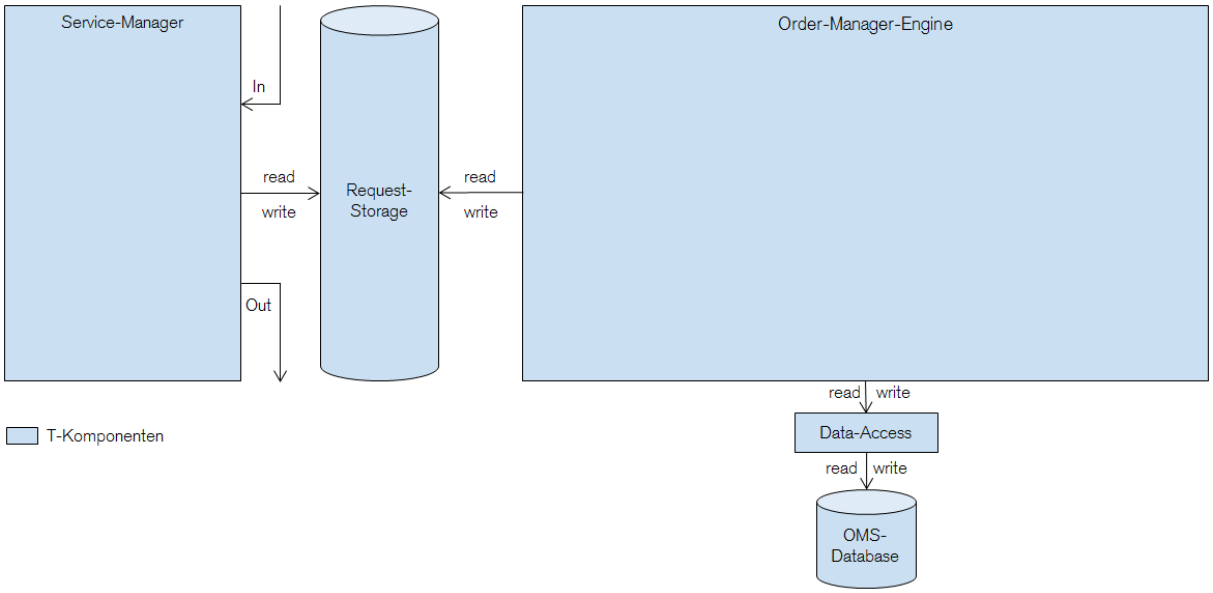


Abbildung 6: Darstellung der T-Komponenten

Abbildung 7 zeigt weitere T-Komponenten der technischen Standardarchitektur. Die Process-Engine stellt die in das OMS eingebettete BPMN 2.0 Engine dar. Diese ist innerhalb der technischen Standardarchitektur fest integriert und ist somit für jede Implementation gleichbleibend. Wie bereits erwähnt besitzt die Process-Engine zur Verwaltung und Ausführung der modellierten BPMN 2.0 Prozesse ein eigenes Datenbank-Schema. Diese Datenbank wird dabei nur von der Process-Engine selbst verwendet, wobei andere Komponenten des OMS keinen Zugriff auf die darin enthaltenen Daten haben. Die verwendete Process-Engine

soll zudem von der OME nicht direkt verwendet werden. Dazu ist eine weitere T-Komponente in die OME integriert, der Process-Engine-Wrapper. Dieser bietet der OME alle Funktionalitäten der Process-Engine, welche für die Ausführung von BPMN 2.0 Prozessen aus Sicht des OMS benötigt werden (siehe 4.9.3.5). Diese Architektur, macht das OMS unabhängiger von der eigentlich verwendeten BPM Plattform bzw. Process-Engine. Der Request-Handler regelt alle ein- und ausgehenden Anfragen aus Sicht der OME. Ruft ein Client beispielsweise einen angebotenen Service des SM auf, so wird die Anfrage vom SM in den RS geschrieben. Daraufhin wird der Request-Handler über die eingehende Anfrage informiert. Dieser soll dann alle weiteren Schritte für die Verarbeitung der Service-Anfrage einleiten (siehe 4.8.2). Auch kann der Request-Handler ausgehende Anfragen an externe Systeme in den RS schreiben. Derartige Anfragen werden von einzelnen Aktivitäten einer Prozess-Instanz gestellt (siehe 4.8.2). Der SM wird dann ebenfalls über dieses Ereignis informiert und liest die ausgehende Anfrage aus dem RS. Anschließend wird dann der externe Service durch den SM aufgerufen.

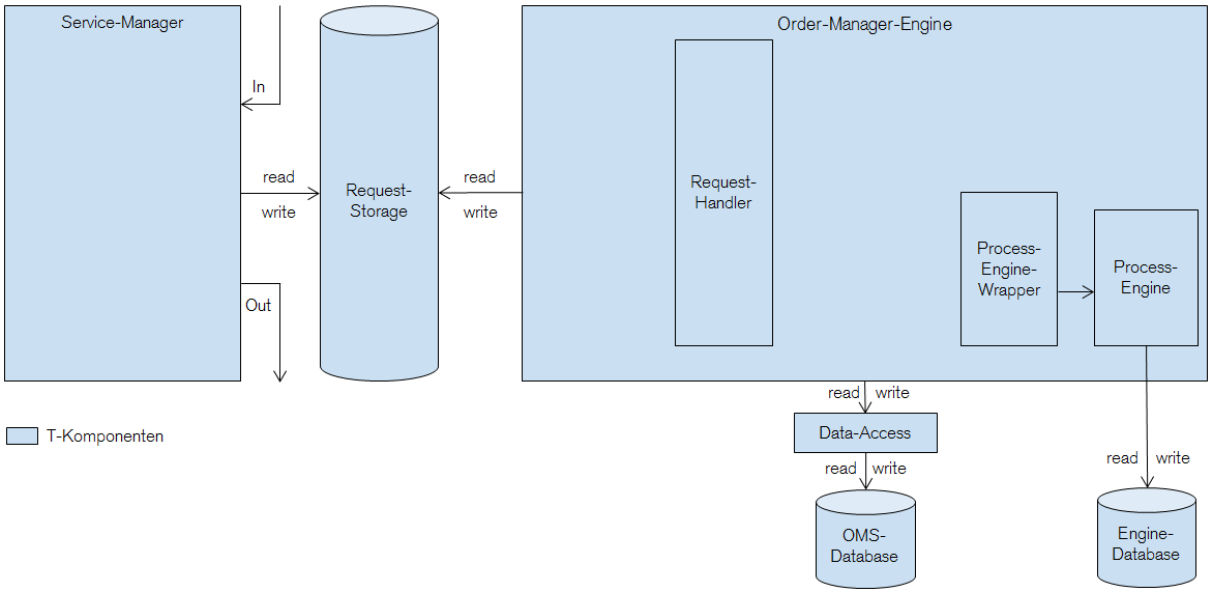


Abbildung 7: Verfeinerte Darstellung der T-Komponenten

4.8 Individuelle Anwendungsarchitektur

Die technische Standardarchitektur aus Kapitel 4.7 dient als Entwurfskonzept für jede konkrete Implementation eines OMS gemäß der technischen Standardarchitektur. Nach Quasar (Siedersleben, 2003) wird bei der Beschreibung einer Anwendungsarchitektur zwischen zwei Kategorien von Komponenten unterschieden. Zum einen gibt es die zuvor beschrieben T-Komponenten (siehe 4.7.1). Zum anderen gibt es anwendungsspezifische Komponenten, welche als A-Komponenten bezeichnet werden. Diese zeichnen sich dadurch aus, dass die-

se für jede Implementation der technischen Standardarchitektur verschieden sind und daher jedes Mal neu definiert und implementiert werden müssen. Zu Beginn werden die A-Komponenten des OMS beschrieben. Anschließend werden diese in Verbindung mit der technischen Standardarchitektur erläutert und grafisch dargestellt.

4.8.1 A-Komponenten

Im Folgenden werden die A-Komponenten des OMS beschrieben. Damit soll ein Überblick über die Komponenten des OMS gegeben werden, welche für jede Implementation eines OMS mit der technischen Standardarchitektur neu bestimmt werden müssen.

Prozess: Für jeden Order-Typ der mit Hilfe des OMS verarbeitet werden soll, muss ein neuer wiederholbarer und aktivitätsorientierter Prozess definiert bzw. modelliert werden. Anschließend muss der Prozess, abhängig eines generischen Prozess-Templates, in einzelne Module unterteilt werden. Im Falle des generischen Prozess-Templates aus 4.5 muss somit jeder Prozess in insgesamt vier Subprozesse unterteilt werden. Kann das Template für den modellierten Order-Typ jedoch nicht verwendet werden, so kann ein neues Template mit Hilfe des neuen Prozesses und dem Order-Life-Cycle aus 2.1 erstellt werden. Die Prozesse befinden sich innerhalb der T-Komponente Order-Manager-Engine und werden von der Prozess-Engine verwendet (siehe Abbildung 8 – „Process Definitions“).

Business-Objekt(e): Für jeden Order-Typ, der mit Hilfe des OMS verarbeitet werden soll, muss ein eigenes Business-Objekt erstellt werden. Aus Sicht des OMS definiert dieses genau die Daten, die in Bezug auf einen bestimmten Order-Typ gespeichert werden sollen. Aus Sicht des Prozesses hingegen, liefert das Business-Objekt alle Daten, welche zur Verarbeitung des Order-Typs notwendig sind. Da der Prozess eines Order-Typs aus mehreren Subprozessen besteht, muss für jeden dieser Subprozesse ebenfalls ein eigenes Business-Objekt erstellt werden. Diskussion Der Grund dafür ist, dass jedem Prozess nur die Daten zur Verfügung stehen dürfen, die zur Verarbeitung benötigt werden (Reichert & Weber, 2012). Somit müssen in Bezug auf einen Prozess, welcher mit Hilfe des generischen Prozess-Templates verarbeitet werden soll, insgesamt fünf Business-Objekte erstellt werden. Ein Objekt für den Order-Typ, sowie jeweils ein Objekt für jeden Subprozess. Die Daten des Business-Objekts eines Subprozesses stellen dabei eine Teilmenge aller Order-Typ spezifischen Daten dar. Dies wird in Kapitel 4.9.3.3 in Bezug auf den entwickelten Prototyp genauer erläutert.

Business-Logik: Ein modellierter Prozess eines Order-Typs besteht aus einzelnen Aktivitäten. Jede dieser Aktivitäten, führt dabei einen logisch zusammenhängenden Arbeitsschritt durch. Um dies zu erreichen, muss der jeweiligen Aktivität, Business-Logik zugewiesen wer-

den. Die Business-logik stellt dann den eigentlichen Anwendungscode dar, welcher zu Verarbeitung der Aktivität ausgeführt werden muss. Dabei muss zwischen interner und externer Business-Logik unterschieden werden. Interne Business-Logik wird vom OMS selbst implementiert und steht den Aktivitäten somit lokal zur Verfügung. Externe Business-Logik hingegen wird von externen Systemen implementiert und dem OMS über Services zur Verfügung gestellt. Soll eine Aktivität also externe Business-Logik verwenden, so muss dafür der entsprechende Service-Aufruf vom OMS implementiert werden. Business-Logik die bereits implementiert wurde, kann von jedem Prozess bzw. jeder Aktivität beliebig oft wiederverwendet werden.

Datenaustauschformat: Der Service-Manager bietet Clients des OMS Services an. Wie bereits erwähnt soll es dabei keine Rolle spielen mittels welcher Technologie diese Services implementiert wurden. Beispielsweise könnte ein Client über RESTful-Services mit dem OMS kommunizieren, während ein anderer Client eine Datei an das OMS sendet. Beide Varianten haben die Aufgabe dem OMS mitzuteilen, welchen Service der jeweilige Client in Anspruch nehmen möchte. Um dies zu realisieren, muss das genaue Datenaustauschformat für jeden dieser Services vorab definiert werden. Somit wird sichergestellt, dass beide Seiten genau wissen, wie genau welche Daten untereinander ausgetauscht werden sollen.

Daten-Mapping: Nachdem das Datenaustauschformat definiert wurde, muss dieses auf die entsprechende Datenformats des OMS gemappt werden. Dies kann aus zwei unterschiedlichen Sichten beschrieben werden. Aus der Sicht des SM, muss eine Service-Anfrage auf das Datenformat des RS gemappt werden. Zusätzlich müssen Anfragen an externe Systeme aus dem RS gelesenen werden können und anschließend in einen entsprechenden Service-Aufruf transformiert werden. Aus Sicht der OME müssen Service-Anfragen aus dem RS gelesen werden können und in das entsprechende Business-Objekt gemappt werden. Auch sollen Service-Anfragen an externe Systeme in den RS geschrieben werden. Dazu muss das Business-Objekt in das Format des RS transformiert werden.

4.8.2 Beschreibung der internen Sicht

Die beschriebenen A-Komponenten sollen nun in die technische Standardarchitektur aus 4.7.3 integriert werden. Dies soll einen Überblick darüber geben, welche Komponenten bei einer Implementierung unterschiedlicher OMS wiederverwendet werden können (T-Komponenten) und welche neu implementiert werden müssen (A-Komponenten).

Abbildung 8 zeigt die in die technische Standardarchitektur integrierten A-Komponenten. Die angebotenen Services des SM stellen das definierte Datenaustauschformat (A-Komponente) dar. Die durch diese Services eingehenden Anfragen werden mit Hilfe des Request-Storage-

Mappers (A-komponente) in den RS geschrieben. Auch werden mit Hilfe des Request-Storage-Mappers, ausgehende Anfragen aus dem RS gelesen. Anschließend werden diese dann für den Aufruf externer Services verwendet.

Die OME enthält sämtliche definierte Prozesse bzw. Subprozesse (A-Komponente) der einzelnen Order-Typen. Diese werden von der Process-Engine zur Verarbeitung der Order-Typen verwendet. Die zu einem Order-Typ gehörenden Business-Objekte (A-Komponente), werden von allen Komponenten der OME verwendet und stellen das interne Datenmodell der OME dar. Der Request-Storage-Mapper (A-Komponente) der OME transformiert eingehende Anfragen in das entsprechende Business-Objekt und schreibt ausgehende Anfragen in den RS. Die ein- und ausgehenden Anfragen werden dabei durch den Request-Handler (T-Komponente) gesteuert. Die Business-Logik (A-Komponente) kann aus zwei unterschiedlichen Sichten betrachtet werden. Aus Sicht eingehender Service-Anfragen, wird die Business-Logik durch den Request-Handler aufgerufen. Diese kann dann über den Process-Engine-Wrapper (T-Komponente) auf die Prozess-Logik des OMS zugreifen. Aus Sicht ausgehender Service-Anfragen wird die Business-Logik durch den Business-Logik-Wrapper (A-Komponente) aufgerufen und leitet die Anfrage dann an den Request-Handler weiter. Der Business-Logik-Wrapper wird von der Process-Engine (T-Komponente) aufgerufen und soll die Business-Logik von der Prozess-Logik trennen. Dieses Vorgehen wird in 4.9.3.6 genauer erläutert.

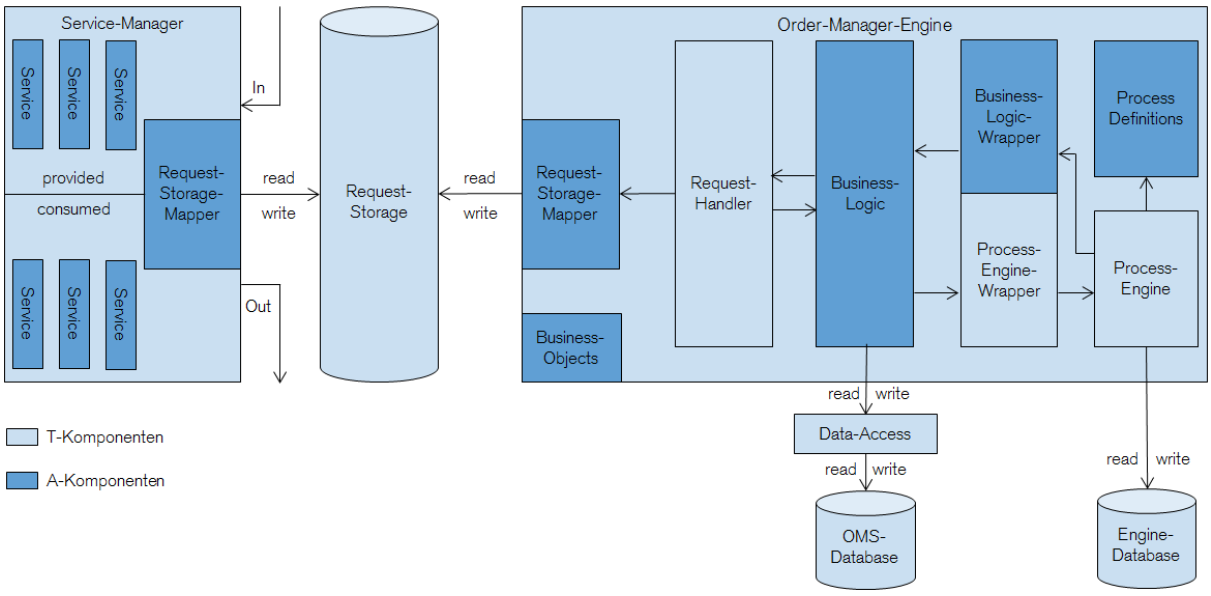


Abbildung 8: Darstellung der A- und T-Komponenten

4.9 Implementierung

In diesem Kapitel wird der Prototyp beschrieben, welcher die in 4.7 gezeigte, technische Standardarchitektur des OMS implementiert. Dabei wird zu Beginn die Entwicklungsumgebung beschrieben. Anschließend wird die individuelle Anwendungsarchitektur mit Hilfe der in 4.8.1 erläuterten A-Komponenten gezeigt. Am Ende dieses Kapitels, werden die wesentlichen Implementationen in Bezug auf die in 3.1 erläuterten Systemanforderungen beschrieben.

4.9.1 Entwicklungsumgebung

Tabelle 1 beschriebt die Entwicklungsumgebung, mit welcher der Prototyp implementiert wurde. Dabei werden die eingesetzten Technologien bzw. verwendeten Frameworks beschrieben und den jeweiligen Komponenten in Bezug auf die Anwendungsarchitektur aus Abbildung 8 zugewiesen. Anhand der Zuweisung lässt sich zudem sehr gut erkennen, wie die einzelnen T-Komponenten der technischen Anwendungsarchitektur technologisch umgesetzt sind.

Art	Name	Version	Komponente(n)
Programmiersprache	Java	Version 7, Update 45	Alle Komponenten
IDE	Eclipse Kepler	-	-
Web Application Server	Apache Tomcat	7.0.50	Servive-Manager
Datenbank Server	MySQL Server	5.6.14	OMS-Database, Process-
Datenbank Server	MongoDB	2.4	Request-Storage
RESTful Services	Jersey	2.5.1	Service-Manager
Workflow- und BPM-Plattform	Activiti	5.14	Order-Manager-Engine
Mapping-Framework	EclipseLink	2.5.1	Data-Access
Mapping-Framework	Spring Data MongoDB	1.3.2	Request-Storage-Mapper
Dependency/Build-Management	Maven	3.1.1	Alle Komponenten

Unit-Testing	JUnit	4.11	Alle Komponenten

Tabelle 1: Entwicklungsumgebung

Für eine modulare Entwicklung, wurde die Organisationsstruktur des Prototyps mit Hilfe von Maven in einzelne Module (Subprojekte) unterteilt. Dazu wurde für die die T-Komponenten Service-Manager, Order-Manager-Engine, Request-Storage und Data-Access jeweils ein eigenes Maven-Modul erzeugt. Um die (Daten)-Objekte des OMS innerhalb allen Modulen verwenden zu können, wurde auch dafür ein Maven-Modul erzeugt.

4.9.2 A-Komponenten

In diesem Abschnitt werden die A-Komponenten des Prototyps beschrieben. Mit Hilfe dieser Beschreibung wird auch festgelegt, welche konkreten Funktionalitäten der Prototyp in Bezug auf die zuvor beschriebene individuelle Anwendungsarchitektur (siehe 4.8) implementiert. Es ist zu beachten, dass mit Hilfe des Prototyps das in dieser Arbeit entwickelte Entwurfskonzept lediglich verifiziert werden soll. Daher wurden zur Implementation einfache und leicht verständliche Beispiele verwendet.

Prozess: Aus zeitlichen Gründen implementiert der im Rahmen dieser Arbeit entwickelte Prototyp nur den Erstellungsprozess einer Order. Abbildung 4 zeigt die modularen Einheiten des FX-Spot Order Beispielsprozesses aus 4.3. Daraus ergibt sich in Bezug auf das Create-Modul des Prozesses der in Abbildung 9 dargestellte Subprozess. Dabei ist zu beachten, dass der Subprozess ein eigenes Start- und End-Event besitzt. Dieser Subprozess kann dann dem Create-Modul des generischen Prozess-Templates (siehe 4.5) zugewiesen werden. Dieses Vorgehen ist in Abschnitt 4.9.3.2 genauer erläutert.

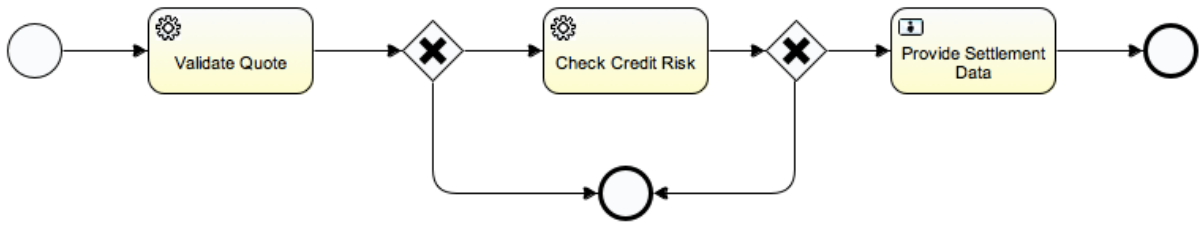


Abbildung 9: FX-Spot Create-Modul Subprozess

Business-Objekt: In Kapitel 4.8.1 wurde beschrieben, welche(s) Business-Objekt(e) für einen jeden Order-Typ erstellt werden müssen. Tabelle 2 zeigt die Daten, welche in Bezug auf die FX-Spot Beispiel Order verarbeitet werden müssen. Der Attributs-Name gibt dabei den eigentlichen Variablennamen an, der Typ definiert den konkreten Datentyp jedes Attributs und die Beschreibung gibt Aufschluss über die jeweilige Funktion des Attributs. Wie zuvor

beschrieben, sollte jeder (Sub)-Prozess nur auf die Daten Zugriff haben, die zu Verarbeitung des (Sub)-Prozesses benötigtet werden. Die Verfügbarkeit gibt daher für jedes Attribut an, innerhalb welchem (Sub)-Prozess ein Attribut Verfügbar sein soll. In Bezug auf den Prototyp, wurde das Business-Objekt absichtlich sehr einfach konzipiert. Der Grund dafür ist, dass eine Verifizierung der Anwendungsarchitektur nicht von der Komplexität der Daten abhängig ist. In Abschnitt 4.9.3.3 wird auf die Verwaltung der einzelnen Business-Objekte eines Order-Typs in Verbindung mit einer laufenden Prozess-Instanz eingegangen.

Attribut-Name	Typ	Beschreibung	Verfügbarkeit
id	Long	Eindeutige Identifikation	Template
orderType	Enum	Der auszuführende Order-Typ	Template
Date	Calendar	Erteilungsdatum der Order	Template
amount	BigDecimal	Gehandelter Betrag	Template
quote	BigDecimal	Wechselkurs des Währungspaars	Template
currencyToBuy	String	Zu kaufende Währung	Template
currencyToSell	String	Zu verkaufende Währung	Template
orderStatus	Enum	Aktueller Orderstatus im Life-Cycle	Template, Create-Modul
validQuote	Boolean	Gibt an ob der Wechselkurs gültig	Template, Create-Modul
credible	Boolean	Gibt die Kreditwürdigkeit des Kunden	Template, Create-Modul
settlementData	String	Auftragsabwicklungs-Daten	Template, Create-Modul

Tabelle 2: Business-Objekt - FX-Spot Order Beispielprozess

Business-Logik: Für den Erstellungsprozess einer FX-Spot-Order, muss der hier entwickelte Prototyp die Business-Logik für jede der drei in Abbildung 9 dargestellten Aktivitäten implementieren. Dabei wird wie in Kapitel 4.9.2 beschrieben, zwischen interner und externer Business-Logik unterschieden. Die Aktivität „Validate Quote“ stellt eine Prüfung des übergebenen Währungskurses dar und wird als interne Business-Logik implementiert. „Check Credit Risk“ stellt die Prüfung der Kreditwürdigkeit des Kunden dar. Diese Prüfung wird durch ein

externes System durchgeführt und ist demnach als externe Business-Logik implementiert. Der Service-Aufruf wird dabei von dem Prototypen lediglich simuliert. Abschließend wird im Erstellungsprozess der FX-Spot Order ein User-Task, „Provide Settlement Data“ ausgeführt. Dabei soll ein an dem Prozess beteiligter Benutzer, Daten zu Abwicklung der späteren Zahlung zur Verfügung stellen. Da ein solcher User-Task würde in einem Produktivsystem mit Hilfe eines Worklist-Systems (siehe 2) verarbeitet werden.

Aus Sicht der Prozess-Logik des OMS, wird beim Aufruf von Business-Logik nicht zwischen interner und externer Business-Logik unterschieden. Wird während der Verarbeitung einer Prozess-Instanz Business-Logik aufgerufen, so weiß das OMS selbst ob die jeweilige Logik intern oder extern implementiert ist (siehe 4.9.3.6). Der Prototyp soll mit Hilfe der drei Aktivitäten zeigen, wie der Aufruf interner und externer Business-Logik implementiert werden kann.

Datenaustauschformat: Das Datenaustauschformat definiert die angebotenen CRUD-Services des OMS (siehe 4.8.1). Im Rahmen des Prototypen wird dazu lediglich der Create-Service zum Erteilen einer Order implementiert. Der Service wird dabei als RESTful-Service implementiert und über den Service-Manager zur Verfügung gestellt. Tabelle 3 zeigt die grundlegende Definition des Create-Services. Dieser ist innerhalb des Prototyps mit Hilfe von Jersey implementiert (siehe 4.9.1).

URL	HTTP-Methode	Übergabeparameter
Serveraddress:port/sevice_manager/rest/fxspot/post	PUT	amout:double, quote:double, currencyToBuy:String, currencyToSell:String

Tabelle 3: Create-Service Definition

Daten-Mapping: In Kapitel 4.8.1 wurde die Funktionalität des Daten-Mappings erläutert. Dieses muss in Bezug auf das für den Prototyp definierte Business-Objekt, sowie das Datenaustauschformat implementiert werden. Um Daten in- und aus dem Request-Storage (MongoDB) mappen zu können, verwendet der Prototyp das Mapping-Framework Spring Data MongoDB (siehe 4.9.1). Mit Hilfe dieses Frameworks, können Objekte einfach und ohne Mapping-Konfigurationen in den Request-Storage geschrieben/gelesen werden.

4.9.3 Erklärung wesentlicher Implementierungen

In diesem Abschnitt wird die grundlegende Implementation von Prozess-Logik unter Verwendung von Activiti erläutert. Dabei wird gezeigt, wie das OMS das generische Prozess-Template einem bestimmten Order-Typ entsprechend konfiguriert, Prozessparameter beim Starten von neuen Prozess-Instanzen übergibt und den Life-Cycle-Status einer Order verwaltet. Außerdem wird erläutert, wie genau das Aufrufen von Prozess- und Business-Logik implementiert ist.

4.9.3.1 Das generische Prozess-Template

Durch das generische Prozess-Template aus Abbildung 5 ist es möglich, verschiedene Order-Typen auf dieselbe Weise zu verarbeiten. Dies gilt für alle Prozesse, welche sich in die vier Module Create, Mature, Delete und Replace unterteilen lassen (siehe Abbildung 4). Wird beispielsweise eine neue Anfrage für eine FX-Spot Order gestellt, so wird zu Beginn eine neue Instanz des Prozess-Templates gestartet. Die jeweilige Instanz wird dann der Anfrage, d.h. dem Order-Typ entsprechend, konfiguriert. Wie in 4.1.3 erklärt, besitzt jede Prozess-Instanz ein eigenes (Daten)-Objekt. Dieses Datenobjekt wird bei der Instanziierung eines neuen Prozesses als Parameter übergeben. Während der Ausführung einer Instanz, wird auch der Status einer Order verändert. Diese Information ist in dem jeweiligem (Daten)-Objekt gespeichert. In den folgenden Abschnitten, werden diese Funktionalitäten in Bezug auf das Create-Modul des generischen Templates (Abbildung 5) sowie des in Abbildung 9 dargestellten Erstellungsprozesses einer FX-Spot Order, veranschaulicht.

In BPMN 2.0 stellt das Create-Modul, einen Subprozess dar und wird als eine sogenannte „callActivity“ modelliert. Eine „callActivity“, ruft dann einen in einer eigenen Datei modellierten Subprozess auf. Abbildung 10 zeigt das Create-Modul des generischen Prozess-Templates in der zugehörigen XML Schreibweise. Dabei besitzt die „callActivity“ für das Create-Modul des generischen Templates insgesamt drei Attribute. Die „id“ stellt eine eindeutige Identifikation des Moduls dar, während das Attribut „name“ der namentlichen Beschreibung dient. Das Attribut „calledElement“ ist die „id“ des wiederholbaren Subprozesses, welcher durch die „Call Activiti“ ausgeführt werden soll. Des Weiteren können mit Hilfe der Attribute „activiti:in“ und „activiti:out“, Ein- und Ausgabeparameter des Subprozesses festgelegt werden. Der Execution Listener hat die Aufgabe auf das Ende der Verarbeitung der „callActiviti“, also des Subprozesses, zu warten. Ist das Ende der Verarbeitung eines Subprozesses erreicht, so wird die Klasse OrderStateManager (siehe Attribut „class“) instanziiert und die Statusänderung der jeweiligen Order eingeleitet (hier „ACTIVE“).

```
<callActivity id="createModule" name="Create"
calledElement="${templateCreateModule}">
      <extensionElements>
            <activiti:in sourceExpression="${createModuleData}"
            target="createModuleData" />
            <activiti:out source="${createModuleData}"
            target="createModuleData">
            <activiti:executionListener
            class="oms.engine.business.lifecycle.OrderStateManager"
            event="end">
                  <activiti:field name="newOrderState"
                  stringValue="ACTIVE" />
            </activiti:executionListener>
      </extensionElements>
</callActivity>
```

Abbildung 10: Generisches Prozess-Template - Create Modul

4.9.3.2 Konfiguration des generischen Prozess-Templates

Die „Call Activity“ des Templates aus Abbildung 10 soll den FX-Spot Create-Modul Subprozess aus Abbildung 9 ausführen. Dazu muss dem Attribut „calledElement“ die „id“ des Subprozesses zugewiesen werden. In Activiti wird dies mit Hilfe von sogenannten „Expressions“ realisiert. Diese bieten die Möglichkeit, auf Prozessparameter einer laufenden Prozess-Instanz zuzugreifen und werden mit der Schreibweise ${parametername} angegeben. Abbildung 10 zeigt die „Expression“ ${templateCreateModule}. Der Wert des Parameters stellt dann die „id“ des aufzuführenden Subprozesses dar (hier FX-Spot Create Modul Subprozess). In Bezug auf den Prototyp ist das Übergeben dieses Parameters für jedes Modul des Templates obligatorisch. Folglich wird beim Starten einer neuen Prozess-Instanz des Templates für jedes Modul ein Parameter mit der „id“ des entsprechenden Subprozesses übergeben. Somit kann das Template mit Hilfe von „Expressions“, einem Order-Typ entsprechend konfiguriert werden. Um dies zu erreichen, muss für jeden Order-Typ definiert werden, welche Subprozesse für welches Template-Modul ausgeführt werden sollen.

4.9.3.3 Prozessparameter

Wird eine neue Prozess-Instanz des Templates gestartet, muss gleichzeitig ein zugehöriges (Daten)-Objekt existieren (siehe 4.1.3). Dieses Objekt wird dann beim Starten des Prozesses als Parameter übergeben. Das Objekt enthält dabei alle Daten, welche zur Verarbeitung der Order durch das OMS benötigt werden. Beispielsweise wird mit Hilfe des Objektes beim Starten einer neuen Instanz des Templates angegeben, welcher konkrete Order-Typ verarbeitet werden soll (z.B. FX-Spot Order). Somit kann das Template der Anfrage entsprechend konfiguriert werden. Jeder Instanz eines Prozesses sollen nur die Parameter übermittelt werden, welche zur Verarbeitung des Prozesses notwendig sind (Reichert & Weber, 2012).

In Bezug auf eine Prozess-Instanz des Templates, werden im Laufe der Verarbeitung insgesamt fünf Prozesse durchlaufen. Der Template-Prozess selbst, welcher mit Hilfe des (Daten)-Objektes einem Order-Typ entsprechend konfiguriert wurde, sowie ein zugehöriger Subprozess für jedes der vier Module des Templates. Diese Prozesse bilden zusammen den Gesamtprozess der Order ab. Die Ein- und Ausgabeparameter eines jeden Moduls werden dabei unter Verwendung der in Abbildung 10 dargestellten Schreibweise „activiti:in“ und „activiti:out“ definiert.

Beim Starten einer neuen Instanz des Prozess-Templates gibt es somit vier weitere obligatorische Parameter, welche jeweils die Eingabeparameter eines jeden Moduls darstellen. Wenn ein Modul des Templates ausgeführt werden soll, so übergibt die Template-Instanz dem Modul eine Kopie des entsprechenden Parameters. Nachdem das Modul verarbeitet ist, wird der originale Parameter der Template-Instanz mit der jeweiligen Kopie des Moduls überschrieben. Auf diese Weise besitzt jedes Modul (Subprozess), wie auch die jeweilige Instanz des Templates, ein eigenes (Daten)-Objekt. Abbildung 11 stellt diese Funktionsweise grafisch dar.

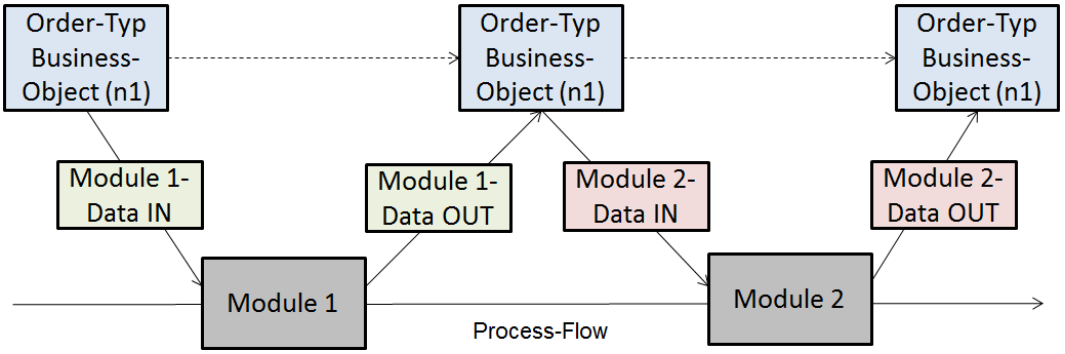


Abbildung 11: Ein- und Ausgabeparameter von Modulen (Subprozessen)

In Tabelle 2 sind alle Parameter aufgelistet, die beim Starten einer Instanz des generischen Prozess-Templates mit Hilfe der Prozess-Engine übergeben werden müssen.

Parametername	Beschreibung
dataObject	Das (Daten)-Objekt für eine Instanz des Prozess-Templates.
templateCreateModule	Die id des Subprozesses für das Create-Modul des Templates.
templateMatureModule	Die id des Subprozesses für das Mature-Modul des Templates.

templateDeleteModule	Die id des Subprozesses für das Delete-Modul des Templates.
templateReplaceModule	Die id des Subprozesses für das Replace-Modul des Templates.
createModuleData	Entsprechendes (Daten)-Objekt des Create-Moduls.
matureModuleData	Entsprechendes (Daten)-Objekt des Mature-Moduls.
deleteModuleData	Entsprechendes (Daten)-Objekt des Delete-Moduls.
replaceModuleData	Entsprechendes (Daten)-Objekt des Replace-Moduls.

Tabelle 4: Parameter einer Instanz des generischen Prozess-Templates

4.9.3.4 Ändern des Order-Life-Cycle-Staus

Wie bereits erwähnt, wird der aktuelle Status einer jeden Order während der Verarbeitung verändert. Der Status ist dabei innerhalb des (Daten)-Objektes gespeichert, welches zusammen mit einer neuen Instanz des generischen Prozess-Templates erzeugt wird. Die Änderung des Orderstatus, erfolgt dabei jeweils nach Abschluss eines jeden Moduls des Templates, wobei diese Funktion während der Modellierung des Templates fest integriert wird (siehe 4.6.1). Abbildung 10 zeigt das Create-Modul des generischen Prozess-Templates in der XML Repräsentation. Der darin enthaltene Execution-Listener hat die Aufgabe, auf das Eintreten des End-Events des Create-Moduls zu warten, was durch das Attribut „event“ definiert ist. Dieses Event wird von der Prozess-Engine beim Abschließen des Subprozesses ausgelöst. Anschließend wird ein Objekt der Klasse instanziiert, welche mit Hilfe des Attributs „class“ angegeben wird. Das Element „activitiField“ des Execution-Listeners gibt unter Verwendung der Attribute „name“ und „string Value“ den neuen Orderstatus an. Das instanzierte Objekt der Klasse kann nun auf diese Variable zugreifen und den jeweiligen Statuswechsel der Order durchführen. Für weitere Informationen bezüglich dieser Funktionalität, wird an dieser Stelle auf den Activiti User Guide verwiesen (Activiti™, 2014).

4.9.3.5 Aufruf von Prozess-Logik

Die Order-Manager-Engine, soll unabhängig von der verwendeten BPM Plattform funktionieren. Dazu stellt die Business-Logik innerhalb der OME ein Interface zur Verfügung. Dieses definiert alle Funktionen, die aus Sicht des OMSs von einer beliebigen BPM Plattform bzw. Prozess-Engine benötigt wird. Im Falle des Prototyps, wurde dieses Interface, unter Verwendung von Activiti implementiert. Es ist denkbar, dass eine alternative BPM Plattform durch

das OMS verwendet werden soll. Dazu muss dem OMS eine konkrete Implementation des Interfaces in Bezug die alternative BPM Plattform zur Verfügung gestellt werden.

4.9.3.6 Aufruf von Business-Logik

Einzelne Prozessaktivitäten, welche innerhalb der (Sub)-Prozesse modelliert sind, nutzen die vom OMS bereitgestellte Business-Logik um die jeweilige Funktion erfüllen zu können. Die Business-Logik soll dabei jedoch nur indirekt von Prozessaktivitäten aufgerufen werden. Der hier entwickelte Prototyp, implementiert dazu für jede Aktivität eines (Sub)-Prozesses eine eigene Java-Klasse. Jede dieser Klassen implementiert dazu ein Activit-spezifisches Interface, welches wiederum die Implementation einer bestimmten Methode erzwingt. Innerhalb dieser Methode wird dann die Business-Logik aufgerufen, welche zur Erledigung der Aktivität benötigt wird. Für eine detaillierte Beschreibung dieser Funktionalität, wird an dieser Stelle auf den Activiti User Guide verwiesen (Activiti™, 2014). Es ist zu beachten, dass diese Vorgehensweise Activiti-spezifisch ist. Beim Einsatz einer alternativen BPM Plattform kann diese daher von der hier gezeigten Lösung abweichen. Grundsätzlich gilt jedoch, dass die vom OMS bereitgestellte Business-Logik nur indirekt aufgerufen werden soll. Für weitere Informationen bezüglich dieser Funktionalität, wird an dieser Stelle auf den Activiti User Guide verwiesen (Activiti™, 2014).

4.10 Endergebnis

Im Folgenden wird das Ergebnis dieser Arbeit in Bezug auf die wissenschaftliche Fragestellung vorgestellt. Zu Beginn wird die Umsetzung der in Kapitel 3.1 erläuterten Systemanforderungen in Bezug auf die Anwendungsarchitektur aus Sicht des Prototyps erläutert und die wissenschaftliche Fragestellung beantwortet. Anschließend wird das Ergebnis kritisch gewürdigt.

4.10.1 Verifikation

In diesem Abschnitt wird die Umsetzung der Systemanforderungen anhand des entwickelten Architekturkonzepts verifiziert. Am Ende steht dann die Antwort auf die wissenschaftliche Fragestellung.

Einsatz einer integrierten BPMN 2.0 Process-Engine: Innerhalb dieser Arbeit wurde für die Modellierung, Verwaltung und Ausführung von BPMN 2.0 Geschäftsprozessen eine OpenSource Workflow- und BPM Plattform mit dem Namen Activiti eingesetzt (siehe 4.2). Der Grund dafür war eine geringere Komplexität bezüglich der Implementierung eines Prototyps zur Validierung der Anwendungsarchitektur. Der Kern der Activiti Plattform, die Prozess-Engine, wurde dabei während der Durchführung der Arbeit in die Anwendungsarchitektur

integriert. Abbildung 8 stellt die Process-Engine als T-Komponente, d.h. als festen Bestandteil der technischen Anwendungsarchitektur, dar (siehe 4.7.1). In Bezug auf den Prototyp und die Architektur bedeutet das, dass die Activiti Prozess-Engine fest in das OMS integriert ist.

Architekturentwurf basierend auf einer Service-Orientierten-Architektur(SOA): Abbildung 8 zeigt die Service-Manager Komponente (T-Komponente) des OMS, welche einerseits Services zur Verfügung stellt und andererseits externe Services aufruft (siehe 4.7.1). Der Service-Manager ist dabei fest in die technische Anwendungsarchitektur integriert und ist somit Bestandteil einer jeden Umsetzung des OMS. Die angebotenen Services werden dann über Service-Manager zur Verfügung gestellt. Dabei spielt es zudem keine Rolle mit Hilfe welcher Technologien die Services implementiert werden. Somit kann das OMS auf verschiedene Weisen in eine SOA integriert werden. Der Prototyp implementiert gemäß Disposition (siehe Anhang 1) dabei nur den Erstellungsprozess einer Order, den Create Service. Externe Service-Aufrufe wie z.B. das Anfragen von Kundendaten und das Aufrufen einer zentralen Worklist für Benutzeraktivitäten, werden seitens des Prototyps nicht implementiert. Diese sind in diesem Dokument theoretisch dargelegt.

Ausführung von Prozessaktivitäten durch interne Funktionen: Für die Orderverarbeitung bietet das OMS neben externen Funktionen (Service-Aufrufen), auch interne Funktionen an. In Bezug auf die Anwendungsarchitektur stellen interne Funktionen A-Komponenten dar und werden im Allgemeinen als Business-Logik bezeichnet (siehe 4.8.1). Abbildung 8 stellt zudem grafisch dar, wie die BPMN 2.0 Process-Engine (Activiti) die Business-Logik über den Business-Logik-Wrapper aufrufen kann. In Bezug auf den entwickelten Prototyp wurde dies in 4.9.3.6 erläutert.

Modulare Erweiterbarkeit des Oder Management Systems: Durch das Aufzeigen der Modularität des Order-Life-Cycles (siehe 4.4.1) und des FX-Spot Order Beispielprozesses (siehe 4.4.2), konnte das generische und modulare Prozess-Template entwickelt werden (siehe 4.5). Ein (Order)-Prozess kann somit bei der Modellierung in einzelne Subprozesse (Module) unterteilt werden, wobei dann jeder dieser Subprozesse einem Modul des generischen Prozess-Templates zugewiesen werden kann. Die modulare Erweiterbarkeit des Systems, lässt sich anhand eines Beispiels erläutern. Zwei Order-Typen könnten sich beispielsweise dahingehend unterscheiden, dass diese während des Erstellungsprozesses unterschiedliche Logik implementieren. Der restliche Verarbeitungsprozess der beiden Order-Typen wäre jedoch identisch. Dies würde zur Folge haben, dass für jeden dieser Order-Typen lediglich ein eigenes Create-Modul in Bezug auf das generische Prozess-Template erstellt werden muss. Die

restlichen Module könnten jedoch ohne weiteres wiederverwendet werden, da der Prozess ansonsten identisch ist.

Auch wird durch die Modularisierung der (Order)-Prozesse die Wartbarkeit erhöht. Beispielsweise könnte es sein, dass ein bestimmter Arbeitsschritt in einem bestimmten Modul aufgrund einer regulatorischen Änderung angepasst werden muss. Die Änderung kann somit an einer zentralen Stelle durchgeführt werden und würde sich dann auf alle Order-Typen auswirken, die das entsprechende Modul verwenden.

Parallele Entwicklung neuer Order-Typen: Durch die Modularität von (Order)-Prozessen ist es möglich, dass theoretisch mehrere Entwickler gleichzeitig an der Entwicklung eines Order-Typs arbeiten können. Während ein Entwickler an dem Create-Modul eines Order-Typs arbeitet, könnte ein anderer das Delete-Modul implementieren. Aufgaben lassen sich somit einfacher verteilen, was wiederum Vorteile in Bezug auf Kosten und Zeit mit sich bringt.

Trennung von Prozess- und Anwendungs-Logik: Für den Entwickler soll es einfach sein, Prozess-Logik von Business-Logik zu trennen. Abbildung 8 stellt diesen Sachverhalt grafisch dar. Zu einem wird gezeigt, wie die Process-Engine, Business-Logik aufrufen kann (siehe auch 4.9.3.6). Auch wird gezeigt, wie die Business-Logik, die Process-Engine aufrufen kann (siehe 4.9.3.5). Durch die Trennung von Prozess- und Anwendungs-Logik ist es möglich, fachliche und technische Aspekte eines Prozesses innerhalb derselben Applikation voneinander zu trennen. Aus der praktischen Sicht eines Entwicklers hat dies den Vorteil, dass die Prozess-Logik der Anwendung in einem ersten Schritt auf einer höheren, nicht technischen, Ebene modelliert werden kann. Dabei wird schon vor der Implementierung ein Verständnis für den eigentlichen Programmverlauf generiert. Zusammen mit diesem Grundverständnis wird dann mit der eigentlichen Implementierung des Prozesses begonnen.

Beantwortung der wissenschaftlichen Fragestellung: Für die Implementation/Integration neuer Order-Typen in das OMS, ist es zu Beginn notwendig die T-Komponenten der technischen Standardarchitektur aus 4.7.1 zu implementieren (Initialaufwand). Die T-Komponenten stellen die Komponenten dar, welche für unterschiedliche Implementationen des OMS gleichbleibend sind. Die T-Komponenten in Bezug auf den in dieser Arbeit entwickelten Prototypen sind in Tabelle 1 beschrieben. Um einen neuen Order-Typ in die technische Standardarchitektur zu integrieren, müssen die jeweiligen A-Komponenten abhängig des zu integrierenden Order-Typs definiert werden (siehe 4.8.1). Daraus ergibt sich für jeden Order-Typ ein zusätzlicher „minimaler“ Aufwand. Der Prozess zur Bestimmung der einzelnen A-Komponenten in Bezug auf den Prototypen, ist dabei in Kapitel 4.9.2 beschrieben.

Daraus lässt sich sagen, dass die Einführung neuer Order-Typen in eine bestehende Implementation der technischen Standardarchitektur des OMS, die Definition und Implementierung der A-Komponenten vorrausetzt. Die A-Komponenten hängen dabei immer von dem Order-Typ ab, welcher in das OMS integriert werden soll. Bereits implementierte A-Komponenten wie z.B. Prozess- oder Business-Logik können dabei jederzeit wiederverwendet werden.

Soll die Verarbeitung einer Order gestartet werden, so kann das OMS mit Hilfe der Prozess-Engine das generische Prozess-Template aus Abbildung 5, einem Order-Typ entsprechen konfigurieren (siehe 4.9.3.2). Das bedeutet, dass jedem Modul des Templates das entsprechende Modul (Subprozess) des Order-Typs zugewiesen wird. Daraus ergibt sich dann der Gesamtprozess, welcher zur Verarbeitung der Order ausgeführt wird. Somit können alle Order-Typen, welche innerhalb des OMS implementiert sind, auf die gleiche Art und Weise gestartet und ausgeführt werden. Zudem vereinfacht der Einsatz einer integrierten Prozess-Engine das Starten, Konfigurieren und Verarbeiten von Prozessen. Durch die einheitliche Implementierungsumgebung, müssen durch das OMS zur Kommunikation mit der Prozess-Engine beispielsweise keine Services verwendet werden.

4.10.2 Kritische Würdigung

Das entwickelte Architekturkonzept in Bezug auf die in dieser Arbeit behandelte Anwendungsklasse (Order Management), stellt die Grundlage für ein standardisiertes Entwurfsmodell zur Verfügung. Dabei wurde mit Hilfe der T-Komponenten die Standard Anwendungsarchitektur beschrieben. Diese kann als eine Art Grundbaustein bezeichnet werden, welcher dann von jeder Applikation der Anwendungsklasse verwendet werden kann. Durch das Architekturkonzept wird zudem dargestellt, wie eine Applikation diesen Grundbaustein ihren jeweiligen Anforderungen entsprechend anpassen kann. Dazu muss jede Applikation lediglich die individuellen A-Komponenten definieren. Durch dieses Vorgehen wird der Grundbaustein abhängig der definierten A-Komponenten zu einer individuellen Applikation. Durch das Verwenden einer in das System integrierten Prozess-Engine (Activiti), wird zudem das Ausführen und Verwalten der (Order)-Prozesse vereinfacht. Der Grund dafür ist, dass das OMS zum Zugriff auf die Prozess-Logik keine externen Systeme ansprechen muss. Die Prozess-Logik kann dadurch auf einer natürliche Art und unter Verwendung derselben Programmiersprache verwendet werden.

5 Schlussfolgerungen und Ausblick

Soll ein neuer Order-Typ in das OMS integriert werden, so wird zuallererst der jeweilige Prozessverlauf aus der fachlichen Sicht definiert. Dadurch wird der eigentliche Verlauf der Orderverarbeitung schon früh im Entwicklungsprozess bekannt, wobei auf die damit verbundenen technischen Aspekte vorerst nicht weiter eingegangen wird. Erst nach der Definition des Verarbeitungsverlaufs, wird mit der eigentlichen Implementierung der Anwendungs-Logik begonnen. Ein Programmierer kann sich somit während der Entwicklung anhand des bereits definierten Prozesses orientieren um die technischen Anforderungen des Order-Typs umzusetzen. Der Vorteil bei der Verwendung einer in das OMS integrierten Prozess-Engine ist, dass die fachlichen und technischen Aspekte zusammen in einer Anwendung verwaltet bzw. implementiert werden können. Dies vereinfacht die Entwicklung aus Sicht des Anwendungsentwicklers, da dieser, Prozess- und Anwendungs-Logik innerhalb seiner gewohnten Arbeitsumgebung implementieren und verwalten kann. Aus praktischer Sicht hätte der Einsatz einer zentralen BPM Plattform den Nachteil, dass dem Anwendungsentwickler die Prozess-Logik nur unter Verwendung eines externen Systems zur Verfügung steht. Anstatt die ausgelagerte Prozess-Logik für die Implementierung der Anwendungs-Logik zu verwenden, ist es aus Sicht des Entwicklers oft einfacher auf den Einsatz der zentralen BPM Plattform zu verzichten. Stattdessen würde dieser mit Hilfe der ihm vertrauten Mittel den Verarbeitungsverlauf des Order-Typs implementieren. Somit würde wiederum keine Trennung zwischen Prozess- und Anwendungs-Logik stattfinden.

Für die weiterführende Entwicklung könnte die in dieser Arbeit verwendete Open Source Workflow- und BPM Plattform (Activiti) mit einer kommerziellen Plattform, wie z.B. Oracle BPM Plattform 11g oder auch Amazon Web Services (AWS), ersetzt werden. Dabei könnte auch die Frage der Portierbarkeit untersucht werden. Praktisch würde eine solche Portierung auf ein kommerzielles System einen größeren Funktionsumfang bieten. Somit könnten zukünftige Order Management Systeme den erhöhten Funktionsumfang einer kommerziellen BPM Plattform verwenden und gleichzeitig, von den zuvor erläuterten Vorteilen einer integrierten Prozess-Engine profitieren. Basierend auf dieser Arbeit, könnte die eigentliche Anwendungsarchitektur für den produktiven Betrieb entwickelt werden. Damit könnte die technische Standardarchitektur als eine Standard-Softwarekomponente definiert, entwickelt und für die Implementation unterschiedlicher Order Management Systeme zur Verfügung gestellt werden. Eine solche Software-Komponente, würde somit die Entwicklung der entsprechenden Anwendungsklasse standardisieren und gleichermaßen maßgeblich vereinfachen.

Literaturverzeichnis

Activiti™. (2014). *Activiti Homepage*. Abgerufen am 09. 01 2014 von www.activiti.org: www.activiti.org

Activiti™. (2014). *Activiti User Guide*. Abgerufen am 25. 01 2014 von www.activiti.org: http://www.activiti.org/userguide/

BPM Offensive Berlin. (2011). *BPMN Poster.* Abgerufen am 31. 01 2014 von www.bpmb.de: http://www.bpmb.de/images/BPMN2_0_Poster_DE.pdf

Ould, M. A. (2005). *Business Process Management.* Swindon: British Informatics Society Limited.

Reichert, M., & Weber, B. (2012). *Enabling Flexibility in Process-Aware Information Systems.* Berlin: Springer-Verlag.

Siedersleben, J. (04 2003). Abgerufen am 26. 11 2013 von https://www.fbi.h-da.de: https://www.fbi.h-da.de/fileadmin/personal/b.humm/Publikationen/Siedersleben_-_Quasar_1__sd_m_Brosch_re_.pdf

Signavio. (26. 09 2012). *BPMN 2.0: Prozessautomatisierung mit jBPM und Activiti*. Abgerufen am 31. 01 2014 von www.signavio.com: http://www.signavio.com/de/news/bpmn-2-0-prozessautomatisierung-mit-jbpm-und-activiti/

SVSP. (2014). *Strukturierte Produkte*. Abgerufen am 29. 01 2014 von http://www.svsp-verband.ch: http://www.svsp-verband.ch/home/potenzial.aspx?lang=de

Anhang

Anhang 1: CD mit dem Programm-Code des Prototyps

Total Lines of Code in den Maven Projekten “service_manager”, “engine”; “data_access”, “data”, “request_storage”: 2115

Anhang 2: Speicherorte wichtiger Klassen und Ressourcen des Prototyps

Anhang 3: Disposition

Anhang 2: Speicherorte wichtiger Klassen und Ressourcen des Prototyps

Speicherorte von Klassen und Ressourcen der definierten A-Komponenten aus Kapitel 4.9.2

Prozess-Definitionen:	(*Sub)-Projekt:*	engine
	src-Order:	src/main/ressources
	root-package	*diagrams.templates*
Daten-Objekte/Enums/Konstanten:	(*Sub)-Projekt:*	data
	src-Order:	src/main/java
	root-package	*oms.data*
Business-Logik:	(*Sub)-Projekt:*	engine
	src-Order:	src/main/java
	root-package	*oms.engine.business*
Datenaustauschformat:	(*Sub)-Projekt:*	service_manager
	src-Order:	src/main/java
	root-package:	*oms.service_manager_services*

Speicherorte der an der Templatekonfiguration beteiligten Klassen und Ressourcen

Prozess-Definitionen:	(*Sub)-Projekt:*	engine
	src-Order:	src/main/ressources
	root-package	*diagrams.templates*
Activiti BPM Engine:	(*Sub)-Projekt:*	engine
	src-Order:	src/main/java
	package	*oms.engine.wrapper.bpm.activiti*
	Klasse:	*ActivitiBpmEngine.java*
Konstanten:	(*Sub)-Projekt:*	data
	src-Order:	src/main/java
	package	*oms.data.constants*
	Klasse:	*Constants.java*

Speicherorte der beteiligten Klassen und Ressourcen zur Übergabe von Prozessparametern

Activiti BPM Engine:	(*Sub)-Projekt:*	engine
	src-Order:	src/main/java
	package	*oms.engine.wrapper.bpm.activiti*
	Klasse:	*ActivitiBpmEngine.java*
Konstanten:	(*Sub)-Projekt:*	data
	src-Order:	src/main/java
	package	*oms.data.constants*
	Klasse:	*Constants.java*

Speicherorte der beteiligten Klassen und Ressourcen zur Änderung des Order-Status

Order State Manager:	(*Sub)-Projekt:*	engine
	src-Order:	src/main/java
	package	*oms.engine.business.lifecycle*
	Klasse:	*OrderStateManager.java*

Speicherorte der beteiligten Klassen und Ressourcen zum Aufruf von Prozess-Logik (Process-Logic-Wrapper)

Interface BPM Engine:	(*Sub)-Projekt:*	engine
	src-Order:	src/main/java
	package	*oms.engine.wrapper.bpm*
	Klasse:	*IBpmEngine.java*
Activiti BPM Engine:	(*Sub)-Projekt:*	engine
	src-Order:	src/main/java
	package	*oms.engine.wrapper.bpm.activiti*
	Klasse:	*ActivitiBpmEngine.java*

Speicherorte der beteiligten Klassen und Ressourcen zum Aufruf von Business-Logik (Business-Logic-Wrapper)

Interface BPM Engine:	(*Sub)-Projekt:*	engine
	src-Order:	src/main/java
	root-package	*oms.engine.wrapper.business.task*

Anhang 3: Disposition

CREDIT SUISSE

Disposition zur Bachelor-Thesis

Order Management System

Martin Bröckl, Wirtschaftsinformatik/Softwareengineering, martinbroeckl@googlemail.com

30.10.2013

Inhaltsverzeichnis

Motivation

Mein Interesse für die Softwareentwicklung hat sich schon früh im Studium bemerkbar gemacht, wodurch mir die Wahl der Vertiefungsrichtung Software-Engineering leicht gefallen ist. Dieses Interesse wurde im Laufe des Studiums immer größer und weckt in mir den Wunsch in Zukunft einen Beruf innerhalb dieses Fachgebietes auszuüben. Meine Bachelor-Thesis, welche ich bei der Credit Suisse AG in Zürich schreiben werde, soll ein weiterer Schritt in Richtung dieses Berufsziels sein und mir dabei helfen einen tieferen Einblick in die Praxis der Softwareentwicklung zu erhalten.

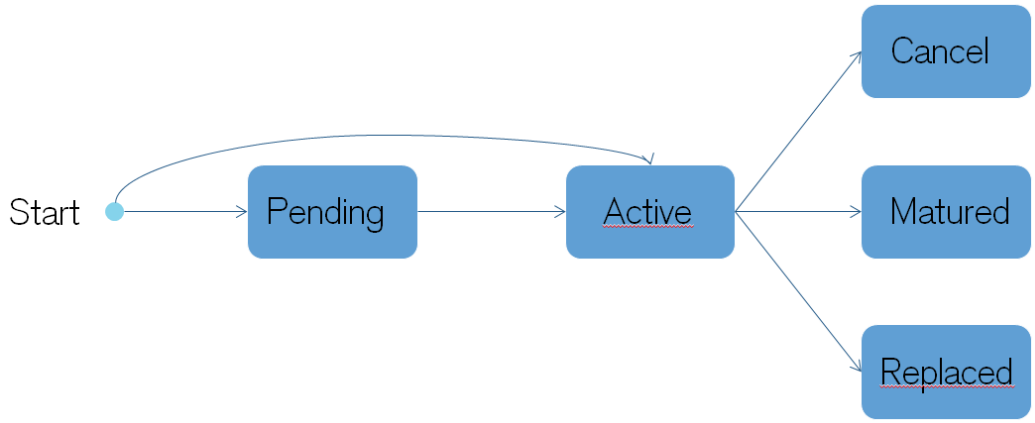
Meine Aufgabe besteht im Allgemeinen darin eine Architektur für ein neues Order Management System zu entwerfen und anschließend einen Prototyp zu implementieren. Unterstützt werde ich dabei von Herrn Dr. Beat Liver von der Credit Suisse, welcher mich von Firmenseite aus betreut und mir bei Fragen bezüglich des Projektes behilflich ist. Von dem betreuenden Professor meiner Thesis wünsche ich mir Unterstützung bei formalen sowie allgemeinen fachlichen Fragen.

1. Problemstellung

Das „Order Management“ der Credit Suisse und in der Finanzindustry umfasst eine Klasse von Anwendungen, um Aufträge seien dies Zahlungsaufträge, Börsenaufträge, Aufträge zum Kauf und Verkauf von außerbörslich gehandelten Titeln, wie Devisen etc., verarbeiten zu können. Da sich die verschiedenen Orders im Allgemeinen einen gemeinsamen Life Cycle teilen, stellt sich die Frage ob es möglich ist einen für diese Anwendungsklasse generischen Order Manager zu entwickeln, der diesen Life Cycle implementiert und es ermöglicht Orders in einem System zu verwalten. In Bezug auf die Thesis, wird diese Fragestellung soweit eingegrenzt werden, dass der damit verbundene Arbeitsaufwand an die Anforderungen einer Bachelor-Thesis angepasst ist.

1.1 Order Life Cycle

Bestandteil jeder Order ist ein für diese Arbeit vorgegebener, allgemeiner Life Cycle, welcher im Folgenden kurz dargestellt und beschrieben ist.



Abblidung 1: Order LifeCycle

- **Pending:** Bedeutet, dass eine Order unvollständig ist und dass noch auf weitere Informationen gewartet wird. Dies können Informationen von anderen Systemen oder aber auch Menschen sein, die an einem Prozess beteiligt sind.
- **Active:** Bedeutet, dass sich eine Order in der Durchführungsphase befindet. Dies kann beispielsweise eine Überweisung von einem auf ein anderes Konto sein.
- **Matured**: Bedeutet, dass die Verarbeitung der Order abgeschlossen ist.
- **Cancel**: Bedeutet, dass die Durchführung der Order gestoppt wurde und alle notwendigen Schritte eingeleitet wurden um die Verarbeitung der Order abzubrechen bzw. rückgängig zu machen.
- **Replaced:** Bedeutet, dass eine Order durch eine oder mehrere Nachfolgeorder(s) ersetzt und die Verarbeitung der aktuellen Order daher beendet ist. Der ursprüngliche Prozessfall wird durch die Verarbeitung der Nachfolgeorder(s) weiter bearbeitet.

1.2 Beispielprozess

Um das Thema für die Thesis eingrenzen zu können, wird ein Beispielprozess aus dem Devisenhandel, eine sogenannte „FX Spot Order“, zu Hilfe genommen. Diese bildet im Wesentlichen den in Punkt 1.1 beschriebenen Life Cycle ab. Bei einer FX Spot Order wird der eigentliche Geldwechsel (Settlement) in der Regel zwei Tage nach Erteilung der Order durchgeführt. Abweichungen davon, werden zur Vereinfachung nicht weiter berücksichtigt.

Ein Kunde der Credit Suisse möchte beispielsweise Schweizer Franken kaufen, wobei die zu verkaufenden Währung Euro ist. An dieser Stelle beginnt die Pre Trade Prozess-Phase, in der der Kunde bei der Bank eine Anfrage für den aktuellen Kurs (Quote) in Bezug auf das gewünschte Währungspaar macht. Nach Erhalt der Quote kann der Kunde entweder die Quote akzeptieren und den Trade abschliessen, oder aber er fragt bei der Bank eine neue Quote an.

Akzeptiert er die Quote, initiiert er damit einen neuen Trade im Order Management System (OMS FX) der Bank. Bevor der Trade jedoch akzeptiert werden kann, prüft das OMS FX ob die Quote zum Abschlusszeitpunkt noch gültig ist (validate quote) und welches Kreditrisiko (check credit risk) die Bank mit dem Kunden und dem Abschluss haben würde (what-if). Im Business-Exception-Fall kann das OMS FX den Trade abbrechen und leitet die dafür notwendigen Schritte ein. Um den Prozess zu vereinfachen, werden an dieser Stelle keine weiteren Schritte berücksichtigt.

Wenn alle Prüfungen erfolgreich waren, kann das OMS FX den Trade akzeptieren. An dieser Stelle wird ein neues Trade Business Objekt erzeugt und es gibt entweder die Möglichkeit den Trade State auf **Pending** oder auf **Active** zu setzen (siehe 1.1). State Änderungen bringen neben einer Änderungen des Objektes noch weiter Schritte mit sich, welche das OMS FX einleiten muss. Datenbankeinträge bzw. Änderungen, lokale oder externe Serviceaufrufe oder aber auch Benachrichtigungen von Usern die an dem Prozess beteiligt sind. Um den Trade später verarbeiten (betrifft auch den credit check) zu können benötigt das OMS FX beispielsweise die Kundendaten (get client data) sowie sie Daten die für die Abwicklung des Geldtransfer notwendig sind (get settlement data). Solange der Trade noch nicht abgeschlossen bzw. **Matured** ist, kann er vom Benutzer storniert **(Cancled)** oder auch geändert **(Update)** werden. Bei gewissen updates wird das aktuelle Trade-Objekt gesichert und ein neues Objekt (und nicht nur eine neue Version) erstellt, welches dann einen Verweis auf das ersetzte Objekt erhält. Wird der Trade gecancelt, leitet das OMS FX die notwendigen Schritte ein um den Vorgang sauber abzubrechen.

Wenn der Trade State active ist, dann wird dem Benutzer eine Bestätigung des Geschäfts (Confirmation) zugestellt. Diese Bestätigung wird nicht vom OMS FX erstellt, sondern von einem eigenen Service der Bestätigungen für diverse Produkte und Dienstleistungen erstellt. Dieser Service wird vom OMS FX lediglich aufgerufen. Das OMS FX wartet ab diesem Zeitpunkt zwei Tage bis zur eigentlichen Abwicklung des Geschäfts (Cash Settlement). In diesem Beispiel bedeutet dies, dass die beiden Zahlungen für den Währungstausch ausgeführt werden. Das OMS FX setzt dann den Status der Order auf „Matured“ und informiert am Abwicklungsdatum ein anderes OMS (OMS Payments) welches für den Währungstausch zuständig ist. Dies stellt für das OMS FX ebenfalls nur einen Serviceaufruf dar.

1.3 Abgrenzung des Themas

Die Aufgabenstellung der Thesis soll nun sein, einen Blueprint für eine Architektur zu entwerfen, welche den allgemeinen Order Life Cycle implementiert, Schnittstellen für interne/externe Serviceaufrufe zur Verfügung stellt sowie beteiligten Personen die Interaktion mit Prozessen ermöglicht. Auch soll es möglich sein mit wenig Aufwand, neue Order-Teilprozesse oder Prozess-Varianten in das System zu integrieren. In der Vergangenheit wurden bereits Prototypen für diese Fragestellung implementiert, jedoch gibt es aktuell kein System bei dem die Prozess-Engine in den Web Application Server (WAS) des Order Management Systems integriert ist. Bisher werden derartige Komponenten immer getrennt voneinander ausgeführt. Somit sollen nicht nur Performance-Vorteile erreicht werden, auch soll das System die Implementierung bzw. Integration neuer Prozesse (aus Sicht des Entwicklers) vereinfachen.

In der Thesis werden die folgenden Themen bearbeitet:

- Architekturentwurf des Order Management Systems mit dem Life-Cycle Status basierend auf Service Orientierte Architektur (SOA), d.h. das OMS bietet CRUD Services an und verwendet im Life Cycle, für die Ausführung von Prozessaktivitäten, SOA Services, interne Funktionen (EJB's) oder Benutzerinteraktion
- Implementierung der Trade-Engine, welche den Order Life Cycle für FX Spot ausführt jedoch ohne Update, Delete und Matured. Diese sind im Architekturentwurf nur theoretisch dargelegt. Folglich wird der Prototyp nur die Erfassung einer Order (Create) implementieren wobei die delegierten Services nicht implementiert bzw. simuliert werden. Dabei soll die Architektur des System so aufgebaut sein, dass die CRUD-Services theoretisch parallel zueinander implementiert werden können, was zudem die Modularität des Systems fördert.
- Implementierung des Systems unter Verwendung einer BPMN 2.0 Process-Engine (Activiti) welche in das System eingebettet werden soll.
- **Optional:** Der End-To-End Prozess wird als Monitoring Workflow modelliert, womit die Gesamtausführung überwacht werden kann. Dies ist notwendig, da die geforderten Fire-and-Forget downstream service invocations nicht direkt quittiert werden. Zusätzlich bietet dies die Möglichkeit den gesamten Prozess darzustellen und somit besser verstehen und testen zu können.
- **Optional:** Nachweis der modularen Erweiterungsfähigkeit durch die Implementation eines weiteren CRUD-Services und des dazugehörigen Prozesssegments. Im einfachsten Fall, könnte das ein einfacher Storno vor der Abwicklung mit Bestätigung umfassen.
- **Optional:** Konkretisierung der konzeptionellen Architektur mit Oracle BPMN gemäß CS Verwendung oder AWS

-

1.4 Anforderungen an das System

Im Folgenden sind die wesentlichen Anforderungen aufgelistet, die an das System gestellt sind.

- Das System soll modular erweiterbar sein, wobei insofern keine technisch, vollwertigen Software-Module definiert werden. Das System lediglich zur besseren Erweiterbarkeit in einzelne Services unterteilt. Erweiterbar bedeutet in diesem Zusammenhang folgendes:
    - Neue Services (lokal, remote) und neue Order-Types (neue Prozesse) können mit einem Minimum an Aufwand in das System integriert werden
    - Services sind versionierbar d.h., es gibt beispielsweise verschiedene Versionen von Update mit unterschiedlichen Aufgaben
    - Bereits vorhandene Prozesse können wohldefiniert geändert und ausführbar gemacht werden
- Die Prozess-Engine ist in das OMS eingebettet (d.h. Workflow-Engine ist in dem WAS mit MDBs und EJB eingebettet).
- Das OMS bietet modulare Services an, um eine Order zu verwalten (CRUD-Services) --> Einbettung in eine Dienste-Orientierte-Architektur (SOA). Diese Services können in Benutzer- und Maschinenschnittstellen verwendet werden.
- Die Architektur soll dabei die Möglichkeit bieten die Services und Order-Typen unabhängig voneinander zu implementieren
- Neue Prozesse (Order-Typen) lassen sich mit „wenig“ Aufwand in das System integrieren
- Separation of Concerns zwischen Prozess, Auftragsdatenobjekt, Geschäftsregeln sowie zwischen generischen Auftragsmanagementfunktionen und spezifischen Funktionen wie z.B. Business Logik. 2. Umsetzung mit Dependency Injection (z.B. JEE CDI), etc..
-

Die Entwicklung des Systems ist mit Hilfe der Folgenden Tools geplant:

- Eclipse Kepler
- Activiti --> Version 5.14
- Spring --> Version 3.2.4
- Datenbank --> MySql Version 1.1 64-bit
- Mapping-Tool --> EclipseLink 2.5.1
- Maven für Dependency-Management --> Version 3.1.1
- Junit für Unit-Testing --> Version 4.11
- JMockit um nicht zu implementierenden Code zu simulieren --> Version 1.5
- Java --> Version 7 Update 45
- JBoss 6.1.0

2. Forschungsfrage

Hauptfrage:

Mit welcher Architektur für ein Order Management System kann eine modulare Erweiterbarkeit, mit Hilfe von wohldefinierten konzeptionellen und/oder programmatischen Schnittstellen, erreicht werden?

Unterfragen:

- Was ist ein Service, eine Order und ein Prozess und wie hängen diese zusammen?
    - Welche modularen Einheiten gibt es?
    - Wie sind diese in den Gesamtprozess für die Orderverarbeitung eingebettet?
- Wie kann die Architektur als Muster (mit Quasar) konzeptionell beschrieben werden?
- Wie kann die Architektur mit der Prozess-Engine Activiti implementiert werden?
- Wie kann die Architektur beschaffen sein? (Modellierung, Validierung mit Prototyp)

3. Methode

Im Folgenden sind die einzelnen Schritte aufgelistet, um die wissenschaftliche Fragestellung zu beantworten.

- Literatur, CS-interne Beispiele – Kondensiert und als Case Study aufbereitet
- Entwicklung einer Architektur unter Berücksichtigung des allgemeinen Life Cycles
- Entwicklung eines Klassendiagramms der einzelnen System-Komponenten
- Entwicklung des Prototypen mit zugehöriger Qualitätssicherung
- Verifizierung/Evaluation der Architektur am Prototypen
- Schreiben des Thesis-Dokuments

4. Vorläufige Gliederung

1. Problem Description
2. Summary of Domain „Financial Services Order Management“
3. Requirements: SOA, … and thesis use case requirements, Architecture Method, Non-f functional requirements
4. Infrastructure Assumptions (preparation phase work)
5. Thesis Architecture
6. Order Management System Implementation
7. Evaluation of Implementation

5. Zeitplan

Zeitraum für die Thesis ist der 11.11.2013 – 11.02.2014

11.11.2013	Start Thesis
18.11.2013	System-Architektur
09.12.2013	Klassendiagramm
23.12.2013	Implementierung
03.02.2014	Evaluierung Implementation
11.02.2014	Thesis Ende